### UCLA UCLA Electronic Theses and Dissertations

#### Title

A Real-Time and Robust Multivariate Estimator for Dynamic Systems with Heavy-Tailed Additive Uncertainties

**Permalink** https://escholarship.org/uc/item/4fd4m19b

**Author** Snyder, Nathaniel Jeffrey

Publication Date 2023

Peer reviewed|Thesis/dissertation

#### UNIVERSITY OF CALIFORNIA

Los Angeles

## A Real-Time and Robust Multivariate Estimator for Dynamic Systems with Heavy-Tailed Additive Uncertainties

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Mechanical Engineering

by

Nathaniel Jeffrey Snyder

2023

© Copyright by Nathaniel Jeffrey Snyder 2023

#### ABSTRACT OF THE DISSERTATION

A Real-Time and Robust Multivariate Estimator for Dynamic Systems with Heavy-Tailed Additive Uncertainties

by

Nathaniel Jeffrey Snyder Doctor of Philosophy in Mechanical Engineering University of California, Los Angeles, 2023 Professor Jason L. Speyer, Chair

In this dissertation, a real-time, analytic and recursive multivariate state-estimation algorithm is developed for time-invariant, time-varying and nonlinear dynamical systems. Unlike Gaussian based state-estimation algorithms, the proposed state-estimation algorithm uses Cauchy random variables to model the uncertainties in the process and measurement functions. For this reason, it is referred to as the multivariate Cauchy estimator (MCE). The MCE uses a characteristic function representation of the conditional probability density function of the system state vector, given the measurement history, which generates the conditional mean and covariance estimates of the system state vector at each estimation step. The characteristic function of the MCE is enhanced in this dissertation from its previous form by an innovative, computationally tractable, and reduced structure. In particular, the backward recursive, or tree-like, evaluation procedure of the previously-used characteristic function is replaced by a linear parameterization. This linear parameterization compresses the backward recursive characteristic function at each estimation step and allows similar terms of the characteristic function to now be combined together, which was previously not possible. Compressing the characteristic function is shown to lead to the elimination of over 99% of terms that previously comprised it after several estimation steps, although the number of terms after a measurement update still grows. Therefore, a method is developed to run the MCE for arbitrary simulation lengths and for the multivariate setting, despite the growing size of the characteristic function. Furthermore, the estimation structure of the MCE is extended to handle nonlinearities in both the system dynamics and the measurement model, in a fashion similar to that of the extended Kalman filter. It is then shown that the MCE algorithm can achieve real-time computational performance by exploiting the parallel structure of the compressed characteristic function, which is done by distributing the computation onto general-purpose graphical processing units. Through several linear and nonlinear dynamic simulations, the MCE and the extended MCE are shown to outperform the Kalman filter and the extended Kalman filter, respectively, for dynamical systems within heavy-tailed noise environments. Monte Carlo experiments illustrate the exciting robustness properties of the proposed estimator over the class of symmetric alpha-stable probability density functions. The dissertation of Nathaniel Jeffrey Snyder is approved.

Robert Thomas M'Closkey Jeffrey D. Eldredge Lieven Vandenberghe Jason L. Speyer, Committe Chair

University of California, Los Angeles 2023 To my loving and supportive family. Thanks for everything.

# Contents

1	Introduction		
	1.1	Motivation	3
	1.2	Symmetric- $\alpha$ -Stable Probability Density Functions	8
	1.3	Least Squares Fitting of Probability Density Functions	9
	1.4	Least Squares Fitting of Characteristic Functions	11
	1.5	Contributions	12
2 Background on the Characteristic Function of the Multivariate (			
	Estimator		16
	2.1	Estimator Formulation	16
		2.1.1 Measurement Update Co-alignment	20
	2.2	Time-Propagation of the Characteristic Function	21
	2.3	Shortcomings of the Backward Recursive Characteristic Function $\ . \ . \ .$	22
3	Compressing the Characteristic Function of the Multivariate Cau		
	Estimator		<b>24</b>
	3.1	Insights on the Backward Recursive Characteristic Function	25
	3.2	Hyperplane Arrangements, Cells, and Cell Enumeration	26
	3.3	A Basis Expansion for Sign-Vectors	31
		3.3.1 Parameterizing the G-Function	32
	3.4	3.4 Equivalence of the Backward Recursive and Compressed Characteristic	
		Functions	33
		3.4.1 Measurement Update at Estimation Step 1 1	33

		3.4.2	Time Propagation at Estimation Step $2 1 \dots \dots \dots \dots \dots$	36
		3.4.3	Measurement Update at Estimation Step $2 2$	37
	3.5	Numer	rical Example of Equivalence	45
	3.6	Term 1	Reduction for the Compressed Characteristic Function	53
	3.7	Dynan	nic Propagation Properties of the Compressed Characteristic Function	54
		3.7.1	Discarding Negligible Terms	56
4	An	Efficie	nt Algorithm for Compressing the G-Function	58
	4.1	Prelim	inaries	58
	4.2	Procee	lure for the Efficient Computation of $\alpha_i^{k k}$	59
		4.2.1	Finding the Coefficients of $ I  = n$	60
		4.2.2	Finding a Point in the Upper Cell of a Vertex	62
		4.2.3	Finding the Coefficients of $ I  < n$	62
		4.2.4	Unpriming the Coefficients of $ I  \le n$	63
		4.2.5	Finding the Coefficients of Arrangements with Degeneracies	64
		4.2.6	Time-Complexity of the Proposed Algorithm	65
		4.2.7	Converting Between the Indicator and Sign Basis	66
		4.2.8	Comments on the Proposed Implementation	67
<b>5</b>	The	Slidin	g Window Approximation	69
	5.1	Deriva	tion of the Sliding Window Approximation	69
		5.1.1	Sliding Window Initialization for Multivariate Systems	70
		5.1.2	Software Architecture of the Sliding Window	76
6	Dist	tribute	d Computation of the Multivariate Cauchy Estimator	77
	6.1	The C	UDA-C Programming Paradigm	77
		6.1.1	Hardware Level Abstraction	78
		6.1.2	Software Level Abstraction	80
	6.2	An Al	gorithmic Cookbook for the Distributed Multivariate Cauchy Esti-	
		mator		83
		6.2.1	Overview	83

		6.2.2	Time-Propagation	83
		6.2.3	Time-Propagation Co-alignment	84
		6.2.4	Measurement Update (Child-Term Generation) $\ldots \ldots \ldots$	86
		6.2.5	G Evaluation	87
		6.2.6	Measurement Update Co-alignment	89
		6.2.7	Alpha Parameterization using Incremental Enumeration $\ . \ . \ .$	90
		6.2.8	Alpha Parameterization using Pinchasi's Method $\ \ldots\ \ldots\ \ldots$	96
		6.2.9	Term Reduction	99
7	$\mathbf{Ext}$	ended	Multivariate Cauchy Estimator for Nonlinear Dynamical Sys	<b>.</b> –
	tem	S		102
8	$\mathbf{Exp}$	erimei	nts	106
	8.1	A Line	ear Three-State Simulation in Cauchy Noise	107
		8.1.1	Formulation	107
		8.1.2	Numerical Results	108
	8.2	A Thr	ee-State Nonlinear Homing Missile Simulation in Heavy-Tailed Noise	e 119
		8.2.1	Formulation	119
		8.2.2	Experimental Simulations and Results	123
	8.3	A Five	e-State Low Earth Orbit Satellite Simulation in Heavy-Tailed Noise	129
		8.3.1	Formulation	129
		8.3.2	Numerical Results	132
9	Con	clusio	ns and Future Work	138
$\mathbf{A}$	App	oendix		142
	A.1	Least	Squares fit of Symmetric- $\alpha$ -Stable Characteristic Functions $\ldots$	142
	A.2	Equati	ing Gauss-Markov to Poisson Telegraph Statistics	145

# List of Figures

1.1	Scalar Cauchy pdf over estimation horizon	7
1.2	Cauchy and Gaussian pdfs	10
1.3	Least squares fit of scalar Cauchy and Gaussian pdfs	11
3.1	Visualization of the cells of hyperplane arrangements	27
3.2	Incremental enumeration visualization	30
5.1	Sliding window approximation	70
6.1	Hardware abstraction of a GPU	78
6.2	Software abstraction of a GPU program	81
6.3	Visualization of GPU driven Incremental Enumeration	92
8.1	Three state Cauchy estimator over eight steps	108
8.2	Three state Cauchy Estimator over a long estimation horizon	115
8.3	Process and measurement noise realization for Fig. 8.2	116
8.4	Close-up of Fig. 8.2	117
8.5	Schematic of the three-state target-pursuer homing missile problem	120
8.6	Process and measurement noise realization for Fig. 8.7	125
8.7	Performance of EKF against Cauchy Estimator for homing missile simulation	126
8.8	Close-up of Fig. 8.7	127
8.9	Monte Carlo averages for homing missile simulation	128
8.10	Measurement and process noise realization for LEO satellite simulation .	133
8.11	Performance of EKF against EMCE for LEO satellite simulation	134
8.12	State history of change in atmospheric density for LEO satellite simulation	134

- 8.13 State history of atmospheric density for LEO satellite in low noise . . . 137
- 8.14 Performance of EKF against EMCE for LEO satellite in low noise . . . . 137

# List of Tables

8.1	Term reduction results	109
8.2	Original Cauchy estimator execution rates	110
8.3	Serial C/C++ execution rates for Cauchy estimator	111
8.4	Performance increase of chapter 4 method over chapter 3 method $\ldots$ .	111
8.5	CUDA-C execution rates for Cauchy estimator	112
8.6	Execution rates using a-priori term reduction	113
8.7	Results of the discarding terms approximation	114
8.8	2,3,4,5 dimensional system execution rates	118

#### Vita

2017	B.S. with Honors in Mechanical Engineering, The Pennsylvania
	State University, Schreyer Honors College
2017-2019	Graduate Research Assistant at the University of California, Los
	Angeles, Electrical and Computer Engineering Department
2019	M.S. in Mechanical and Aerospace Engineering, The University of
	California, Los Angeles
2019-2023	Graduate Research Assistant at the University of California, Los
	Angeles, Mechanical and Aerospace Engineering Department

#### Publications

N. Snyder, M. Idan, and J. L. Speyer, "Distributed computation of a robust estimator based on cauchy noises," in 2021 60th IEEE Conference on Decision and Control (CDC), 2021

N. Snyder, M. Idan, and J. L. Speyer, "Real-time robust multivariate estimator for dynamic systems with heavy-tailed additive uncertainties," in IEEE Journal on Transactions on Automatic Control (under review), 2023

## Chapter 1

## Introduction

The central goal of this dissertation is to present the contributions made to a newly enhanced and now real-time state estimation algorithm for linear and nonlinear dynamical systems, referred to hereafter as the multivariate Cauchy estimator (MCE). To this day, practically all state-of-the-art estimation algorithms use a Gaussian assumption when modeling the statistical uncertainties for both the process and the measurement models for a dynamical system. Although computationally tractable and cheap, the Gaussian probability density function (pdf) does not account well for statistical outliers in its distribution due to the light and exponentially decaying tails of its pdf. If a data-processing algorithm using the Gaussian noise assumption observes a measurement that falls many standard deviations away from its mean, the outlier may not be dealt with appropriately. This shortcoming has necessitated many heuristics to be included within a Gaussian state estimation framework.

Such a circumstance is easily shown when considering the behavior of a Kalman filter that has processed a new measurement that varies greatly from the one prior. The explanation for such deviation is either due to the process making a large and impulsive change in its state or from a large amount of noise entering the sensor acquisition system over the observation period. If the explanation is truly due to a large jump in the underlying system state, the filter may discount such a jump if it is modeled with a Gaussian whose covariance greatly underestimates large deviations. If the explanation is truly due to measurement noise, the net effect of the Kalman gain operating on the measurement residual may cause a large (and false) update to the state estimate. In either case, the filter may need many (statistically 'well-behaved') new observations to re-converge.

The proposed estimation algorithm, the MCE, is different. It explicitly models both the process and measurement uncertainties as additive Cauchy random variables. Cauchy random variables are said to have 'heavy' probability tails, meaning that the decay of its probability curve (from the peak) is sub-exponential. The result is that the Cauchy random variable allots far greater probability towards events which would fall *many* standard deviations away from the mean of a Gaussian. This, coupled with the extraordinary fact the (analytic and recursive) structure of the MCE's underlying pdf can form multimodal hypotheses of the state allows this estimator to quickly hedge between the state hypotheses, or 'beliefs'. Such a mathematical structure will be shown to be particularly useful for environments in which the statistical uncertainty of either the process or the measurements of a system are heavy-tailed, (i.e, volatile or impulsive), more so than a Gaussian pdf would suggest. For statistical uncertainties that are seen to have welldefined and almost Gaussian behavior, followed by periods where large outliers in the data are observed, the MCE is exceptionally robust.

This chapter first formally motivates in section 1.1 the need for robust and real-time state estimators for linear and non-linear dynamic systems operating in heavy-tailed noise environments. The Cauchy, Gaussian, and class of symmetric alpha stable  $(S-\alpha-S)$  pdfs (i.e., 'heavy-tailed' pdfs), along with their characteristic functions are then introduced in section 1.2. This is followed by a discussion on how one can statistically equate the two distributions, or to a third (heavy-tailed  $S-\alpha-S$ ) distribution in sections 1.3 and 1.4. This 'fitting' of pdfs is necessary when conducting the heavy-tailed experiments in chapter 8 between the family of Kalman filtering algorithms and the proposed MCE. The contributions provided by this dissertation are then formally enumerated in section 1.5.

#### 1.1 Motivation

Phenomena of the world does not always behave Gaussian. It has been well recognized that reliance on the Gaussian pdf can be dangerous, since noise data in many practical systems in engineering, economics, biology, financial movements, earthquakes, atmospheric turbulence, etc., are poorly described by Gaussian pdfs and can be captured better by heavy-tailed ones [1]. To better deal with statistically volatile and impulsive data, heuristics have traditionally been incorporated into algorithms that wish to maintain the Gaussian assumption. Such examples include research efforts to 'robustify' the Kalman filter in some manner to make the Gaussian assumption less brittle towards heavy-tailed data, or by altogether replacing the underlying density, for example, by a student's-t distribution [2].

Nevertheless, for linear dynamic systems with additive Gaussian noises, the Kalman filter (KF) along with the linear-exponential-Gaussian (LEG) filters have been the main estimation paradigms in the fields of economics, finance, and engineering [3, 4]. Variants of the KF, such as the extended Kalman filter (EKF), unscented Kalman filter (UKF), particle filter (PF), and Gaussian mixture models (GMM) have all been proposed to either deal with possible nonlinearities in the dynamics, multi-modal state hypothesis, or large environmental uncertainties [5]. In general, these estimation structures have gained popular appeal due to their efficacy and off-the-shelf implementations.

Although the KF, LEG, EKF, UKF, PF, GMM algorithms possess their own distinct advantages, state estimation in the presence of system nonlinearities and heavy-tailed uncertainties is still very challenging. For systems with linear dynamics, the KF and LEG algorithms may greatly underestimate the presence of large statistical outliers. One could tune the process and measurement noise covariances to accommodate an observed 'worstcase' deviation, however, the performance within the interim periods of time will suffer greatly. While the EKF is able to accommodate nonlinear dynamics and measurement models, it can be susceptible to divergence and failure when linearizing around a poor state estimate when large outliers are observed [5]. Moreover, it is well known the state uncertainty due to the linearization step is underestimated. Filters such as the UKF avoid the pitfalls associated with linearization by sampling a set of aptly chosen points, which captures the current system mean and covariance estimates well. The UKF is preferred to the EKF when the system dynamics are highly nonlinear, however, the algorithm will nevertheless suffer when the system and environment noises are not Gaussian.

Particle filters can, however, be a universal approximator to any pdf. Through a sampling procedure provided by a proposal distribution and subsequently a particle weighting step (the likelihood of the observation, given the particle state), the particle set as a whole can maintain multi-modal state hypotheses. Therefore, the particle filter holds a substantial robustness advantage over the algorithms previously listed. However, to represent multiple beliefs that can arise from nonlinear dynamics within a heavy-tailed noise environment, the particle set will require many particles to appropriately estimate the posterior distribution. Moreover, the curse of dimensionality makes this estimator highly unusable when the system state becomes large. Unlike the aforementioned algorithms, real-time performance is now a centrally important issue, as one must maintain a sufficiently large set of particles to determine the posterior density while meeting the real-time system performance requirements<sup>1</sup>. A good comparison of several of the aforementioned algorithms can be found in [7], which illustrates their performance issues while operating within a heavy-tailed noise environment.

Formally, the Gaussian probability density function (pdf) belongs to the larger family of *alpha-stable* probability distributions. Alpha stable distributions are defined, generally, through their characteristic functions and by four parameters of fit: the stability parameter  $\alpha \in (0, 2]$ , skewness parameter  $\beta \in (0, 1]$ , location parameter  $\mu \in (-\infty, \infty)$ , and scale parameter  $\sigma \in (0, \infty)$  [8]. For what follows, our scope will be limited to the class of estimators that naturally arise from the family of symmetric-alpha-stable (S- $\alpha$ -S) distributions for  $\alpha = 1, 2$ , where the distribution is known (Cauchy and Gaussian, respectively) and said to be symmetric if  $\beta = 0$ . Distributions with  $\alpha = 2$  are said to

<sup>&</sup>lt;sup>1</sup>There is a litany of design decisions one would need to consider when constructing a particle filter for robust estimation in non-Gaussian environments. Decisions include choosing a proposal distribution to sample from, importance resampling strategies, mitigating particle depletion, actively changing the size of the particle set, injection of random particles for robustness, etc. However, these issues are not enumerated here. See [6] for a comprehensive overview.

possess well-defined mean and variance, whereas distributions whose  $\alpha < 2$  have infinite variance, or heavy tails.

Many naturally occurring phenomena can be modeled more appropriately by heavytailed distributions than a Gaussian would permit. For example, distance and bearing measurements can be estimated using an active radar in clutter, where the time series data fits an S- $\alpha$ -S probability density of  $\alpha = 1.7$  well [9]. From private conversations, radar in the presence of jamming, clutter, and scintillation noises has even been observed to follow distributions with a  $S - \alpha - S$  value of  $\alpha$  as low as 1.1. In [10], it is observed that the disturbances observed from sonar sensors used in bearings-only-tracking are heavy-tailed and non-Gaussian. In [11], it is further seen that air turbulence emits data distributions with heavier tails than a Gaussian distribution would suggest. In [12], upper atmospheric density data was seen to exhibit characteristics inconsistent with the Gaussian model, and a drag coefficient estimation problem for low earth orbiting satellites benefited from employing a heavy-tailed Bayesian estimation scheme. For data that is consistent with  $\alpha \in (1,2)$ , heavy-tailed modeling is (in part) troublesome, as there exists no closed-form S- $\alpha$ -S probability density function (although the characteristic function does exist). Due to the lack of an analytic pdf for S- $\alpha$ -S values of  $\alpha \in (1,2)$ , closed-form algorithms do not exist for dynamically generating a conditional probability density function (cpdf) in these environments.

Although the (unconditional) Cauchy pdf has infinite variance and mean, it is shown in [13] that the cpdf exists and exhibits a finite mean and variance, expressed in a closed form. While few, if any, real-world processes are truly Cauchy distributed, because the tails of this distribution over-bounds those of realistic physical phenomena, it is hypothesized that estimators based on Cauchy densities will be robust to the unknown physical densities in question [14]. In [14, 15], it was shown that a multivariate estimator based on the Cauchy pdf performed superior to that of the KF in a Cauchy noise simulation and performed very similarly in a Gaussian noise simulation. The original multivariate Cauchy estimator (MCE) of [16] is closed form, analytic, and recursively updates the characteristic function of the unnormalized conditional probability density function (ucpdf) of the system-state vector given the measurement history, at each estimation time step. Notably, the estimator is capable of analytically hypothesizing multi-modal beliefs. This is unlike the particle filter, which discretely hypothesizes its beliefs through the particle set.

The ability to (analytically) hedge between multiple beliefs is motivated in Fig. 1.1, as was first depicted in [13]. The behavior of the estimator between measurement indices 54 and 55 and then between 59 and 60 are particularly noteworthy. At index 54, both a small process noise and a large measurement noise deviation were realized, respectively. We see that the estimator wagers the observation is likely due to process noise, and adjusts its mean estimate accordingly. However, the pdf enlarges its one standard deviation confidence bound greatly to account for the volatility in the observed measurement. We see that, unlike the Kalman filter, the conditional variance of the estimate is indeed a *function* of the measurement history. Now at step 55, the measurement noise and process noise are both much smaller. The Cauchy estimator realized its wager was overconfident, and immediately shifts the pdf mass close to that at index 53. There, it significantly reduces its one standard deviation confidence bound.

Later, at index 59, a very large process noise and a small measurement noise are realized, respectfully. The behavior of the estimator, now, is surprising. The estimator constructs a primary belief the state is in a similar vicinity as the previous state estimate, wagering the observation is likely due to measurement noise. However, it also sets up a *second* belief that the system state could have actually jumped due to the process noise. The estimator is now hedging between the two-state hypotheses of where the system state could be through constructing a bimodal pdf and a large one standard deviation confidence bound. At index 60, the estimator sees the new measurement is consistent with its second belief. The probability density is adjusted accordingly, turning its previously bimodal belief into a unimodal belief centered around its second mode. We see the confidence bound additionally tightens dramatically, and the state estimate is now very close to the true state. The findings from this experiment are that the Cauchy estimator can 1) dynamically adjust its covariance based on the measurement history, 2)



Figure 1.1: Top: Pdf of the scalar Cauchy estimator in a Cauchy noise simulation. Bottom: Process and measurement noise realization.

hypothesize multiple beliefs of the state, and 3) quickly hedge between them.

The original characteristic function of the MCE in [16], however, presented major challenges for achieving a real-time implementation. The characteristic function was seen to be composed of many terms at each estimation step. Each term would generate factorially many new child terms at the successive estimation step. Although it was seen that a large number of these terms were similar, they could not be combined together. This was due to the fact that the terms of this characteristic function were themselves dependent upon all past terms generated at each estimation time step (and thus past characteristic functions). This is to say, the characteristic function generated a backward recursive 'tree-like' structure, where each newly generated child term was a function of all past parent terms and their associated parameters. Therefore, the backward recursive structure of the characteristic function causes two main issues. The first is memory burden. All terms of past estimation steps along with terms of the current estimate step must be held in computer memory. The second issue is that similar child terms could not be combined together. It was seen that many of these terms existed in the characteristic function. This was further exacerbated by the factorial growth rate of terms after a measurement update, producing even more similar terms at future estimation steps. Together, the computation and memory burden of the estimator, unfortunately, was immense after only several estimation steps. The body of work presented in this dissertation focuses on resolving many of the aforementioned computational and memory issues of the characteristic function of [16].

#### **1.2** Symmetric-α-Stable Probability Density Functions

The pdfs and characteristic functions used in this dissertation are presented here. The focus of this work surrounds estimators that arise from members of the  $S-\alpha-S$  family of distributions. Distributions of this family can be expressed generally through their characteristic function as

$$\phi_X(\nu;\alpha,\sigma,\mu) = \int_{-\infty}^{\infty} f_X(x) e^{j\nu x} dx = e^{j\nu\mu + |\sigma\nu|^{\alpha}} \in \mathbb{C},$$
(1.1)

where the real-valued and scalar  $S - \alpha - S$  random variable X has realization x and a pdf denoted as  $f_X(x)$ . Above, j is the imaginary number. The values  $\mu \in \mathbb{R}$ ,  $\sigma \in (0, \infty)$ ,  $\alpha \in$ (0, 2] parameterize the characteristic function. The location parameter  $\mu$  describes where values of x are centered, the scaling parameter  $\sigma$  describes the width of the distribution, and the stability parameter  $\alpha$  describes the decay rate of the tails of the pdf [17].

Of particular interest is when  $\alpha$  is set to a value of either 1 or 2. When  $\alpha = 1$ , the Cauchy pdf is recovered as

$$f_X(x;\mu,\sigma) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \phi_X(\nu;1,\sigma,\mu) e^{-j\nu x} d\nu = \frac{1}{\pi} \left[ \frac{\sigma}{(x-\mu)^2 + \sigma^2} \right],$$
 (1.2)

where both the mean and second moment of the Cauchy random variable are infinite, i.e.,  $E[x] = \infty$ ,  $E[x^2] = \infty$ , respectively. In (1.2),  $\mu$  is the median parameter of the Cauchy pdf and indicates where the peak of the Cauchy pdf is located. Above,  $\sigma$  is the scaling parameter. When  $\alpha = 2$ , the Gaussian pdf is recovered as

$$f_X(x;\mu,\sigma) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \phi_X(\nu;2,\frac{\sigma}{\sqrt{2}},\mu) e^{-j\nu x} d\nu = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$
(1.3)

where the mean and variance are given as  $E[x] = \mu$ ,  $E[(x - \mu)^2] = \sigma^2$ , respectively. Note in (1.3) the Gaussian characteristic function is defined with  $\frac{\sigma}{\sqrt{2}}$ . The multivariate Gaussian pdf is easily found by writing (1.1) in its multivariate form for  $\alpha = 2$  and recovered as

$$f_X(x;\bar{x},P) = \frac{1}{(2\pi)^{\frac{n}{2}}\sqrt{\det(P)}} e^{-\frac{1}{2}(x-\bar{x})^T P^{-1}(x-\bar{x})},$$
(1.4)

where  $x \in \mathbb{R}^n$  denotes the *n*-dimensional Gaussian random vector of mean  $\bar{x} \in \mathbb{R}^n$  and covariance  $P \in \mathbb{S}^{n \times n}_{++}$ .  $\mathbb{S}_{++}$  denotes that P is symmetric positive definite.

Depicted in Fig. 1.2 are zero-centered Cauchy and Gaussian pdfs ( $\alpha = 1, 2$ , respectively) overlayed for different  $\sigma$  values. It is clear the Cauchy pdf allots larger probability in the tails of its pdf when compared to the Gaussian distribution. Even at values of  $\pm 4$ , the tails of either Cauchy pdfs have significantly more probability density than the depicted Gaussians.

## 1.3 Least Squares Fitting of Probability Density Functions

To compare estimators that are based on either Gaussian or Cauchy noise models, the underlying Gaussian and Cauchy pdfs will need to be fitted to one another in a statistical sense [14, 15, 18]. That is, if the process or measurement noises are simulated as Gaussian, an appropriate Cauchy pdf must be fitted to this Gaussian pdf in order to use the MCE. Conversely, for noises that are simulated as Cauchy, an appropriate Gaussian pdf must be fitted to the Cauchy pdf in order to use a Kalman filter.



Figure 1.2: Gaussian and Cauchy pdfs with scaling parameters  $\sigma = \{1.0, 0.5\}$  overlayed, respectively.

The Cauchy and Gaussian distributions can be fitted to one another in the least squares sense by minimizing

$$\underset{\sigma_P}{\operatorname{argmin}} \int_{-\infty}^{\infty} \left( f_{X_T}(x;\mu_T,\sigma_T) - f_{X_P}(x;\mu_P,\sigma_P) \right)^2 dx, \tag{1.5}$$

where  $f_{X_P}(x; \mu_P, \sigma_P)$  is the proposal pdf to be fitted, with  $\sigma_P$  being the optimization variable in question (i.e, the scale parameter of the proposal pdf), and  $f_{X_T}(x; \mu_T, \sigma_T)$ represents the target pdf to fit to. Note that the optimization only takes place over the variable  $\sigma_P$  and not  $\mu_P$ . If the target distribution is not zero-centered (i.e.,  $\mu_T \neq 0$ ), then  $\mu_P$  can simply be set equal to  $\mu_T$  since both distributions are single-peaked and symmetric.

When the target pdf is the Gaussian and the proposal pdf is the Cauchy, the optimization for  $\sigma_P$  yields a ratio of  $\frac{\sigma_P}{\sigma_T} = 0.7195$ . This indicates the Cauchy scaling parameter which minimizes (1.5) is the Gaussian value scaled down by 0.7195. If the Cauchy pdf is set to the target pdf and the Gaussian is the proposal pdf, the ratio becomes  $\frac{\sigma_P}{\sigma_T} = \frac{1}{0.7195} = 1.3898$ . Conversely, the Gaussian scaling parameter which minimizes (1.5) is the Cauchy value scaled up. This makes intuitive sense, as the Cauchy pdf has much heavier tails than the Gaussian pdf does. Figure 1.3 shows the result of fitting the Cauchy pdfs in Fig. 1.2 to the Gaussian pdfs. The derivation of (1.5) can be found in appendix A.1.



Figure 1.3: Cauchy pdfs fitted to Gaussian pdfs, respectively, in the least squares sense.

It should be noted here that there are many ways in which one can measure the distance between two distributions. A popular choice is the KL-divergence measure. Unlike the least-squares approach, however, minimizing the KL-divergence measure for a Gaussian fit to a Cauchy pdf does not produce desirable results, and yields an 'optimal' choice of  $\sigma \to \infty$  for the Gaussian scaling parameter. This is due to the heavy tails of the Cauchy pdf. This occurs as the KL metric attempts to give more probability to the light tails of the Gaussian. For this reason (and convenience), least squares fitting was chosen.

#### 1.4 Least Squares Fitting of Characteristic Functions

For noise data that is not Gaussian or Cauchy, but is seen to be something in between (i.e.,  $\alpha \in (1,2)$ ), both the Gaussian and Cauchy pdfs will need to be fit to this third

distribution. Unfortunately, (1.5) cannot be used, as pdfs for  $\alpha \in (1,2)$  do not have analytic and closed-form solutions. However, the characteristic function does exist, as shown by (1.1). Furthermore, using Plancherell's theorem (here for real-valued functions due to  $\mu_{\{P,T\}} = 0$ ) which states that the integral of a squared real-valued function is equal to the integral of its squared frequency spectrum, the problem in (1.5) is equivalent to

$$\underset{\sigma_{P}}{\operatorname{argmin}} \int_{-\infty}^{\infty} \left( \phi_{X_{T}}(\nu; \sigma_{T}, \alpha_{T}) - \phi_{X_{P}}(\nu; \sigma_{P}, \alpha_{P}) \right)^{2} d\nu$$
$$= \underset{\sigma_{P}}{\operatorname{argmin}} \left[ \int_{-\infty}^{\infty} e^{-2|\sigma_{T}\nu|^{\alpha_{T}}} d\nu - 2 \int_{-\infty}^{\infty} e^{-|\sigma_{T}\nu|^{\alpha_{T}} - |\sigma_{P}\nu|^{\alpha_{P}}} d\nu + \int_{-\infty}^{\infty} e^{-2|\sigma_{P}\nu|^{\alpha_{P}}} d\nu \right]. \quad (1.6)$$

The middle integral above (in general) does not have an analytic expression. However, numerical minimization of (1.6) can be applied. Equation (1.6) now allows any two (or more) distributions in the  $S-\alpha-S$  family to be fit to one another, in the least squares sense. Furthermore, sampling from any of the  $S-\alpha-S$  distributions can be done by using the algorithm of [19]. In chapter 8, the methods outlined here and in section 1.3 are used to fit the process and measurement statistics for the MCE/EMCE and the KF/EKF estimation algorithms to its respective environmental noise distribution.

#### **1.5** Contributions

The central contribution of this dissertation is to reformulate the characteristic function of the MCE from [16] into a computationally reduced structure that is capable of real-time multivariate estimation applications. The contributions of this dissertation are organized into its chapters, as follows.

In chapter 2, background on the original MCE algorithm is given. The central computational and memory burden of its structure is then discussed. It is shown next in chapter 3 that the original backward recursive characteristic function of the MCE can be removed and replaced with a 'compressed' representation at each estimation step. The main contribution here is that by viewing several parameters of a single term of the characteristic function instead as the coefficients of hyperplane arrangements, a cell-enumeration method from either [20, 21] can be applied to find the unique set of sign-vectors describing the cells of the arrangement. The sign vectors found by the cell-enumeration algorithm allow for a linear system of equations to be formed. The solution of which is shown to compress the characteristic function. The ability to compress the characteristic function now allows similar terms that meet a special criterion to be reduced together. The result of this enhancement is that after several estimation steps (i.e., around seven to eight measurement updates) more than 99% of the terms previously comprising the characteristic function are removed.

The parameterization derived in chapter 3 is then shown to have a dynamic propagation formula in section 3.7. A related dynamic propagation property was seen in [13] for the scalar Cauchy estimator, but only now has a closed-form dynamic propagation equation been discovered for these parameters in the multivariate case. The newfound parameterization of terms can be used in conjunction with an interesting result presented in [22] for linear time-invariant (LTI) systems. The result of [22] analytically shows that it is a priori known which terms of the characteristic function can reduce together. Now that it is possible to combine terms together by compressing the characteristic function, terms at the known indices of reduction can be combined into a single term simply by adding together their parameterization vectors. This removes the need to search through all terms for reductions, which would require a search that is run-time quadratic. The net result is that for LTI systems, the estimator can be run exceptionally fast. Moreover, for all dynamic systems (time-invariant, time-varying, and nonlinear), it is observed numerically that for many terms of the characteristic function, the norm of the parameterization vector approaches zero after several estimation steps. This fact implies an additional optimization: terms whose parameterizing vector approaches a norm of machine zero can be discarded from the characteristic function. Since the term is discarded, it will not generate any more terms at future estimation steps.

Now that the compressed structure of the characteristic function has been uncovered, chapter 4 develops a significantly more efficient algorithm to construct the parameterization vector of each term. This method is shown to be much more efficient at finding the parameterization than the previous method of running a cell-enumeration algorithm and then a linear system solver. The algorithm proposed uses several insights from [23] regarding functions of hyperplane arrangements that have constant value within their cells.

Formalizing how the MCE will run simulations for long horizons of time is developed in chapter 5. Although the compressed structure of the characteristic function reduces memory consumption and allows for similar terms to be combined, the number of terms after a measurement update will still grow, albeit now more slowly. A method, termed the sliding window approximation, was proposed in [15] for two-state linear systems to cap the growth rate and run the estimation structure continuously with a fixed computation load per estimation step. The method has been enhanced and is now generalized to multivariate systems. This allows the MCE to run for arbitrarily long simulations with a very minor impact on its state-estimation performance.

The next contribution is to distribute the computation of the MCE, discussed in chapter 6. All subroutines required to evaluate its characteristic function have been implemented using parallel computing on a graphics processing unit (GPU). This is acceptable, as all terms of the characteristic function are independent of one another. The CUDA-C programming language [24] was chosen for this implementation. Distributing the computation of the compressed characteristic function has enabled impressive speed increases from the original MCE formulation. To motivate this point, processing the eighth estimation step for a three-state dynamic system once took over twelve hours in Matlab using the original characteristic function. Now, this can be done in approximately 10 milliseconds using the new GPU-driven framework for three-state systems. Using the new sliding window approximation, the MCE can now be run at 100 Hz. For smaller window sizes, the estimator can be run well over 1000 Hz and still drastically outperform the Kalman filter. For LTI systems, the estimator can run even faster.

Chapter 8 illustrates the utility of the compressed MCE for engineering applications through several motivating examples. Slight changes are first made in chapter 7 to the MCE, so nonlinear dynamics and measurement models can be provided, termed the *extended* multivariate Cauchy estimator (EMCE). These accommodations are similar to how the EKF handles nonlinear systems. A formal analysis of the run-time performance of the proposed framework is provided to showcase the speed improvements. Simulations are then provided which demonstrate the performance of the MCE for heavy-tailed noise environments. Notably, a large-scale Monte Carlo experiment illustrates the robustness properties of the EMCE for a nonlinear system over the class of  $S-\alpha-S$  density functions and for a range of  $\alpha$ -stability parameters between 1 and 2.

## Chapter 2

# Background on the Characteristic Function of the Multivariate Cauchy Estimator

This chapter provides background on the original 'backward-recursive' characteristic function of the MCE algorithm, first derived in [16]. This is needed to understand the formulation of the compressed characteristic function, presented in chapter 3. The original MCE is introduced in section 2.1. The MCE has two main steps: a measurement update step and a time propagation step, which are given in sections 2.1 and 2.2, respectively. Section 2.3 enumerates the computational and memory issues of the original characteristic function summarized by this chapter.

#### 2.1 Estimator Formulation

Given the discrete-time linear dynamic system

$$x_{k+1} = \Phi_k x_k + \Gamma_k w_k, \tag{2.1a}$$

$$z_k = H_k x_k + v_k, \tag{2.1b}$$

with  $x_k \in \mathbb{R}^n$  representing the system-state vector at estimation step k, the transition matrix  $\Phi_k \in \mathbb{R}^{n \times n}$ , the process-noise vector of heavy-tailed Cauchy random variables  $w_k \in \mathbb{R}^r$  and the control matrix  $\Gamma_k \in \mathbb{R}^{n \times r}$ . The measurement vector  $z_k \in \mathbb{R}^p$  is modelled with  $H_k \in \mathbb{R}^{p \times n}$  and Cauchy noise  $v_k \in \mathbb{R}^p$ . Here,  $\Phi_k, \Gamma_k, H_k$  can either be time-invariant or time-varying. The vectors  $x_1, w_k, v_k$  are modeled as independent and heavy-tailed realizations of Cauchy random variables.

The goal of the estimation problem is to determine the conditional mean and covariance of the system state-vector  $X_k$  given the measurement history  $Y_k = \{Z_1, ..., Z_k\}$  at estimation step k. Note that  $X_k, Y_k, Z_k$  are used to represent the random vectors while  $x_k, y_k, z_k$  represent their realizations. The cpdf of the system state-vector at step k given the measurement history realization  $y_k = \{z_1, z_2, ..., z_k\}$  can be written generally as

$$f_{X_k|Y_k}(x_k|y_k) = \frac{f_{X_k,Y_k}(x_k,y_k)}{f_{Y_k}(y_k)} = \frac{f_{Z_k|X_k,Y_{k-1}}(z_k|x_k,y_{k-1})f_{X_k|Y_{k-1}}(x_k|y_{k-1})f_{Y_{k-1}}(y_{k-1})}{f_{Y_k}(y_k)}.$$
(2.2)

Since the propagation of the cpdf (2.2) for scalar linear systems with additive Cauchy noises could not be extended to multivariate systems, it was shown in [16], however, that the characteristic function of the unnormalized conditional probability density function (ucpdf) of the multivariate system indeed exists. The characteristic function of the ucpdf is initialized at k = 1 as

$$\bar{\phi}_{X_1}(\nu) = \exp\left[\left(-\sum_{\ell=1}^n p_\ell^1 \left|\langle a_\ell^1, \nu \rangle\right|\right) + j\langle b^1, \nu \rangle\right]$$
(2.3)

where  $\nu \in \mathbb{R}^n$  is the spectral vector. The parameter  $p_{\ell}^1$ ,  $\ell = [1, ..., n]$  is the scaling parameter of the Cauchy pdf. This is used to model the initial uncertainty of the corresponding system state  $x_{1,\ell}$ . The parameter  $b_{\ell}^1, \ell \in [1, ..., n]$  is the median value (location parameter) for the Cauchy pdf. This parameter describes where in space the corresponding initial state  $x_{1,\ell}$  is likely located. The parameters  $a_{\ell}^1 \in \mathbb{R}^n$ ,  $\ell \in [1, ..., n]$  are chosen as orthogonal directions. Note that the condition  $Ha_{i\ell}^1 \neq 0$  must hold for all indices  $\ell \in [1, ..., n]$ , and the directions  $a_{\ell}^1$  should be chosen to uphold this condition. After selecting  $b^1, p^1, a_l^1$ , the MCE algorithm then processes the first measurement  $z_1 \in \mathbb{R}$  (i.e, the measurement update step). As is shown in [16], the characteristic function of the ucpdf can be expressed generally at any estimation step k after processing the k-th sensor measurement  $z_k$  as

$$\bar{\phi}_{X_k|Y_k}(\nu) = \int_{-\infty}^{\infty} f_{X_k,Y_k}(x_k, y_k) e^{j\nu^T x_k} dx_k$$
$$= \sum_{i=1}^{N_t^{k|k}} g_i^{k|k} \left( y_{gi}^{k|k}(\nu) \right) \exp\left( y_{ei}^{k|k}(\nu) \right).$$
(2.4)

The superscript k|k indicates an estimate at step k uses all the measurements up to  $z_k$ . The characteristic function is expressed as a sum of  $N_t^{k|k}$  terms and starts with a single term at initialization (i.e., the parameters  $b^1, p^1, a_l^1$ ). The subscript 'i' denotes a particular term  $i \in [1, ..., N_t^{k|k}]$ . Each term i is a product between complex-valued functions of the spectral vector  $\nu$ , given by

$$g_{i}^{k|k}\left(y_{gi}^{k|k}(\nu)\right) = \frac{1}{2\pi} \left[ \frac{g_{r_{i}^{k|k}}^{k-1}\left(y_{gi1}^{k|k}(\nu) + h_{i}^{k|k}\right)}{jc_{i}^{k|k} + d_{i}^{k|k} + y_{gi2}^{k|k}(\nu)} - \frac{g_{r_{i}^{k|k}}^{k-1}\left(y_{gi1}^{k|k}(\nu) - h_{i}^{k|k}\right)}{jc_{i}^{k|k} - d_{i}^{k|k} + y_{gi2}^{k|k}(\nu)} \right], \quad (2.5)$$

where

$$y_{gi1}^{k|k}(\nu) = \left(\varrho_i^{k|k}\right)^T \lambda_i^{k|k}(\nu) \in \mathbb{R}^{k-1}, \ \varrho_i^{k|k} \in \mathbb{R}^{(k-1) \times m_i^{k|k}}$$
(2.6a)

$$y_{gi2}^{k|k}(\nu) = \left(q_i^{k|k}\right)^T \lambda_i^{k|k}(\nu) \in \mathbb{R}, \ q_i^{k|k} \in \mathbb{R}^{m_i^{k|k}}$$
(2.6b)

$$\lambda_{il}^{k|k}(\nu) = \operatorname{sgn}\left(\left\langle a_{il}^{k|k}, \nu \right\rangle\right), \ l \in [1, ..., m_i^{k|k}], \ \lambda_i^{k|k}(\nu) \in \mathbb{R}^{m_i^{k|k}},$$
(2.6c)

and an exponent of

$$y_{ei}^{k|k}(\nu) = -\sum_{l=1}^{m_i^{k|k}} p_{il}^{k|k} \left| \left\langle a_{il}^{k|k}, \nu \right\rangle \right| + j \left\langle b_i^{k|k}, \nu \right\rangle \in \mathbb{C}.$$
(2.6d)

The normalization factor  $f_{Y_k}(y_k)$  of (2.2) and (2.4) is given by

$$f_{Y_k}(y_k) = \bar{\phi}_{X_k|Y_k}(\nu)\Big|_{\nu=0} = \sum_{i=1}^{N_t^{k|k}} g_i^{k|k}\left(y_{gi}^{k|k}(\nu)\right)\Big|_{\nu=0^n} \in \mathbb{R}.$$
 (2.7)

In (2.5),  $c_i^{k|k}, d_i^{k|k}, y_{gi2}^{k|k} \in \mathbb{R}$  and the vector arguments  $q_i^{k|k}, \lambda_i^{k|k}(\nu)$  are the parameters generated at step k|k after processing a sensor measurement and  $m_i^{k|k}$  is the number of elements in the vector parameters of term i at k|k. The subscript  $\ell \in [1, ..., m_i^{k|k}]$  is used to reference one of the  $m_i^{k|k}$  elements. The parameters  $y_{gi1}^{k|k}(\nu), h_i^{k|k}, \in \mathbb{R}^{k-1}$  are the arguments to the recursive numerator function  $g_{r_i^{k|k}}^{k-1|k-1}(\cdot) \in \mathbb{C}$  and the parameter  $\varrho_i^{k|k}$  contains updated  $q_i^{(\cdot)}$  parameters (at step k) from past characteristic functions at the time steps 1 through k-1, stored as a 2-D vector. Note that at  $k = 1, g_{r_i^{1|1}}(\cdot) = 1$ . Equation (2.5) is at the heart of the computational and memory challenges of the characteristic function due to  $g_{r_{k}^{k|k}}^{k-1|k-1}(\cdot)$ . It is a recursive function of all parameters of past characteristic functions generated at time steps 1, ..., k-1. The subscript  $r_i^{k|k}$  indicates the parameters held in  $\varrho_i^{k|k}$  and generated by past characteristic functions are updated at k|k and provide the arguments to those past characteristic functions of steps k-1|k-1 recurring back to step 1. Due to the hierarchy of divisions effectively created by  $g_{r_i^{k|k}}^{k-1|k-1}(\cdot)$ , the characteristic function at the current step k|k requires the parameters of all past time steps to be stored in memory. This relation between the parameters of steps 1, ..., k resembles a 'tree-like' structure. For simplicity, the parameters  $y_{gi1}^{k|k}(\nu)$  and  $y_{gi2}^{k|k}(\nu)$  of (2.6a) and (2.6b) can be thought of as functions of the sign-vectors  $\lambda_i^{k|k}(\nu)$  of (2.6c) and the spectral vector  $\nu$ . In (2.6d),  $y_{ei}^{k|k}(\nu)$  is a function of parameters  $p_{il} \in \mathbb{R}, a_{il}^{k|k}, b_i^{k|k} \in \mathbb{R}^n$ , forming a sum over the absolute value of the inner products of vectors  $a_{il}^{k|k}$  and  $\nu$  and then weighted by  $p_{il}^{k|k}$ . The imaginary part is formed through the inner product of  $\nu$  and  $b_i^{k|k}$ . Both  $\lambda_i^{k|k}(\nu)$  and  $a_{il}^{k|k}$  are central to the discussion of chapter 3. The derivation of the functional forms of the parameters  $a_{il}^{k|k}, b_i^{k|k}, c_i^{k|k}, d_i^{k|k}, p_i^{k|k}, q_i^{k|k}, \rho_i^{k|k}, h_i^{k|k}$  can be found in [16], but will also be presented in section 3.4 when comparing the compressed form of the characteristic function in detail to the form presented above.

The conditional mean  $\hat{x}_k$  and estimation error covariance  $P_k$  can be constructed after

the measurement update step from the first and second derivatives, respectively, of the characteristic function evaluated at  $\nu = \epsilon \bar{\nu}$ ,  $\epsilon \to 0$  for some  $\bar{\nu} \neq 0^n$ . They are given by

$$\hat{x}_{k} = \frac{1}{jf_{Y_{k}}} \sum_{i=1}^{N_{t}^{k|k}} g_{i}^{k|k}(\bar{\nu}) \bar{y}_{ei}^{k|k}(\bar{\nu}) \in \mathbb{R}^{d},$$
(2.8)

$$P_{k} = -\frac{1}{f_{Y_{k}}} \sum_{i=1}^{N_{t}^{k|k}} g_{i}^{k|k}(\bar{\nu}) \bar{y}_{ei}^{k|k}(\bar{\nu}) \left(\bar{y}_{ei}^{k|k}(\bar{\nu})\right)^{T} - \hat{x}_{k} \hat{x}_{k}^{T} \in \mathbb{R}^{d \times d},$$
(2.9)

with

$$\bar{y}_{ei}^{k|k}(\bar{\nu}) = -\sum_{l=1}^{m_i^{k|k}} p_{il}^{k|k} \lambda_{il}^{k|k}(\bar{\nu}) a_{il}^{k|k} + j b_i^{k|k} \in \mathbb{C}^d,$$
(2.10)

and  $f_{Y_k}$  is evaluated as in (2.7) while replacing  $\nu = 0^n$  with  $\bar{\nu}$ . For minor restrictions that apply to  $\bar{\nu}$  and for a detailed derivation of (2.8) to (2.10), see [16].

#### 2.1.1 Measurement Update Co-alignment

It was observed in [16] that it is possible for two or more directions  $a_{ij}^{k|k}, a_{i\ell}^{k|k}, \forall j < \ell, j, \ell \in [1, ..., m_i^{k|k}]$  for a term  $i \in [1, ..., N_t^{k|k}]$  to be 'co-aligned', which can be defined as

$$\frac{\left(a_{ij}^{k|k}\right)^{T}a_{i\ell}^{k|k}}{\|a_{ij}^{k|k}\|_{2}\|a_{i\ell}^{k|k}\|_{2}} = \pm 1,$$
(2.11)

This is problematic, as forming the new parameters  $a_{i(\cdot)}^{k+1|k+1}$  will be ill-defined (see section 3.4 or [16] for the measurement update form of  $a_{i(\cdot)}^{k|k}$ ). To remedy this situation, for all parameters  $a_{ij}^{k|k}$ ,  $a_{i\ell}^{k|k}$  for  $j < \ell$ ,  $j, \ell \in [1, ..., m_i^{k|k}]$  of a term *i* which satisfies (2.11), the parameter  $a_{i\ell}^{k|k}$  at index  $\ell$  is removed. Then, the parameter  $p_{ij}^{k|k}$ ,  $q_{ij}^{k|k}$  of index *j* can be updated updated as

$$p_{ij}^{k|k} \leftarrow p_{ij}^{k|k} + p_{i\ell}^{k|k} \operatorname{abs} \left( \|a_{i\ell}^{k|k}\|_2 \right)$$
(2.12a)

$$q_{ij}^{k|k} \leftarrow q_{ij}^{k|k} + q_{i\ell}^{k|k} \operatorname{sgn}\left(\left(a_{ij}^{k|k}\right)^T a_{i\ell}^{k|k}\right).$$
(2.12b)

This property has important ramifications for the compressed characteristic function when deriving its form in chapter 3. After a measurement update and the co-alignment step, the number of terms in the characteristic function becomes

$$N_t^{k|k} = \sum_{i=1}^{N_t^{k|k-1}} (m_i^{k|k-1} + 1), \qquad (2.13)$$

$$m_i^{k|k} \le m_i^{k|k-1} \le m_i^{k-1|k-1} + r.$$
 (2.14)

Equation (2.13) illustrates that the characteristic function has a factorial growth rate with respect to the number of terms  $N_t^{k|k}$ . The relation  $m_i^{k|k} \leq m_i^{k|k-1}$  of (2.14) is due to the co-alignment property presented above, where r in the upper bound is the number of elements in term i added after the time propagation step (and is equal to the number of process noises in  $w_k$  of (2.1), more on this next).

#### 2.2 Time-Propagation of the Characteristic Function

To propagate the MCE from estimation step k to k + 1, the time propagation step is applied. Again, these expressions will be useful when showing the equivalence of the compressed characteristic function in section 3.4 to the forms presented here. The form of the time-propagated characteristic function is given as,

$$\bar{\phi}_{X_{k+1}|Y_k}(\nu) = \sum_{i=1}^{N_t^{k|k}} g_i^{k+1|k} \left( y_{gi}^{k+1|k}(\nu) \right) \exp\left( y_{ei}^{k+1|k}(\nu) \right).$$
(2.15)

where

$$g_{i}^{k+1|k}\left(y_{gi}^{k+1|k}(\nu)\right) = \frac{1}{2\pi} \left[ \frac{g_{r_{i}^{k+1|k}}^{k-1|k-1}\left(y_{gi1}^{k+1|k}(\nu) + h_{i}^{k|k}\right)}{jc_{i}^{k|k} + d_{i}^{k|k} + y_{gi2}^{k+1|k}(\nu)} - \frac{g_{r_{i}^{k+1|k}}^{k-1|k-1}\left(y_{gi1}^{k+1|k}(\nu) - h_{i}^{k|k}\right)}{jc_{i}^{k|k} - d_{i}^{k|k} + y_{gi2}^{k+1|k}(\nu)} \right],$$

$$(2.16)$$

and

$$y_{gi1}^{k+1|k}(\nu) = \left(\varrho_i^{k+1|k}\right)^T \lambda_i^{k+1|k}(\nu) \in \mathbb{R}^k, \ \varrho_i^{k+1|k} \in \mathbb{R}^{k \times m_i^{k+1|k}}$$
(2.17a)

$$y_{gi2}^{k+1|k}(\nu) = \left(q_i^{k+1|k}\right)^T \lambda_i^{k+1|k}(\nu) \in \mathbb{R}, \ q_i^{k+1|k} \in \mathbb{R}^{m_i^{k+1|k}}$$
(2.17b)

$$\lambda_{il}^{k+1|k}(\nu) = \text{sgn}\left(\left\langle a_{il}^{k+1|k}, \nu \right\rangle\right), \ l \in [1, ..., m_i^{k+1|k}], \lambda_i^{k+1|k}(\nu) \in \mathbb{R}^{m_i^{k+1|k}}$$
(2.17c)

with an exponent of

$$y_{ei}^{k+1|k}(\nu) = -\sum_{l=1}^{m_i^{k+1|k}} p_{il}^{k+1|k} \left| \left\langle a_{il}^{k+1|k}, \nu \right\rangle \right| + j \left\langle b_i^{k+1|k}, \nu \right\rangle \in \mathbb{C}.$$
 (2.17d)

Equations (2.16) to (2.17c) are similar to those of (2.5) to (2.6c), with several differences. Note that (2.16),  $h^{k+1|k} = h^{k|k}$ ,  $c_i^{k+1|k} = c_i^{k|k}$  and  $d_i^{k+1|k} = d_i^{k|k}$ . The terms  $p_i^{k+1|k}$ ,  $d_{il}^{k+1|k}$ ,  $b_i^{k+1}$  are updated by  $\Phi_k$  in the time-propagation step and it should be mentioned that  $q_i^{k+1|k}$  will be equal to  $p_i^{k+1|k}$  during time-propagation (see [16] for derivation or section 3.4 for further details). The number of terms of the characteristic function does not increase in time-propagation and  $N_t^{k+1|k} = N_t^{k|k}$ . The number of elements in a term, however, increases by at most r (i.e, the number of process noises of (2.1)), and the relation is given by  $m_i^{k|k} \leq m_i^{k+1|k} \leq m_i^{k|k} + r$ . The inequality is due to the chance of co-alignment in the  $a_{il}^{k+1|k}$  parameters during the time-propagation step. If this is seen to occur, the co-alignment rules outlined in (2.12) for  $p_i^{k|k}$  can be applied here to the time propagated  $p_i^{k+1|k}$ .

# 2.3 Shortcomings of the Backward Recursive Characteristic Function

The backward recursive numerator  $g_{r_i^{k|k}}^{k-1|k-1}(\cdot)$  of (2.5) is at the center of the memory and computational issues of the characteristic function presented in this chapter. To evaluate the conditional mean  $\hat{x}_k$  and covariance  $P_k$  as given by (2.8) to (2.9), the backward recursive structure of (2.5) requires the parameter set  $a_{il}^{k|k}, b_i^{k|k}, c_i^{k|k}, d_i^{k|k}, p_i^{k|k}, q_i^{k|k}, \rho_i^{k|k}, h_i^{k|k}$
of all terms from steps 1 to k to be held in computer memory for access. If this numerator was replaced by an expression dependent only on parameters at k, the terms from steps 1 to k-1 could be forgotten. This would save a large amount of memory. Moreover, doing so would allow any similar terms at step k to be reduced together. As each term creates  $m_i^{k|k-1}$  new terms after a measurement update step, the computational savings could be very large. As discovered in [25], there are indeed many terms at each step k that will combine together.

# Chapter 3

# Compressing the Characteristic Function of the Multivariate Cauchy Estimator

The structure of the compressed characteristic function is now presented. Sections 3.1 to 3.3.1 examines and formulates a geometric way to view and solve for several important parameters that comprise the backward recursive characteristic function. Section 3.4 then explicitly uses these insights and derives the compressed characteristic function of the MCE. Section 3.6 outlines the rules for term-combination, which is now possible, as the characteristic function is no longer backwards-recursive. Lastly, section 3.7 shows the newly constructed parameter that compresses the characteristic function has an interesting dynamic propagation property. In the linear system case, this property reveals a large amount of computation in the characteristic function can be moved offline, prior to processing any sensor measurements. Many of the insights given in this chapter were first presented by the author of this dissertation in [14].

# 3.1 Insights on the Backward Recursive Characteristic Function

The reformulation of this chapter starts by examining equation (2.6b)

$$y_{gi2}^{k|k}(\nu) = \left(q_i^{k|k}\right)^T \lambda_i^{k|k}(\nu) \in \mathbb{R}, \ q_i^{k|k} \in \mathbb{R}^{m_i^{k|k}}$$
$$\lambda_{il}^{k|k}(\nu) = \operatorname{sgn}\left(\left\langle a_{il}^{k|k}, \nu \right\rangle\right), \ l \in [1, ..., m_i^{k|k}], \ \lambda_i^{k|k}(\nu) \in \{\pm 1\}^{m_i^{k|k}},$$

which is embedded in the backward-recursive function of (2.5). We see  $y_{gi2}^{k|k}(\nu)$  is a function of sign-functions, where each sign-function takes the inner-product between the parameters  $a_{il}^{k|k} \in \mathbb{R}^n$  and the spectral vector  $\nu \in \mathbb{R}^n$ . Alternatively, the parameter vector  $a_{il}^{k|k}$  can be thought of as the coefficients of a hyperplane in  $\mathbb{R}^n$  with respect to the spectral vector  $\nu$ . The set of all  $l = \left[1, ..., m_i^{k|k}\right]$  hyperplanes  $a_{il}^{k|k}$  could then be viewed as an *arrangement* of hyperplanes, where all  $\nu$  that satisfy  $a_{il}^{k|k}\nu = 0$  would lie on the hyperplane. From this new viewpoint,  $\lambda_{il}^{k|k}(\nu)$  is a function which indicates (with the values  $\pm 1$ ) whether a value of  $\nu$  lies in the positive halfspace  $H_{il}^+ = \{\nu \in \mathbb{R}^n \mid a_{il}^{k|k}\nu > 0\}$  or the negative halfspace  $H_{il}^- = \{\nu \in \mathbb{R}^n \mid a_{il}^{k|k}\nu < 0\}$  of the hyperplane, respectively.

Formally, the set of vectors  $a_{il}^{k|k}, l \in [1, \ldots, m_i^{k|k}]$ , grouped into a matrix  $A_i^{k|k} \in \mathbb{R}^{m_i^{k|k} \times n}$ , defines a *central* arrangement of hyperplanes of dimension  $\mathbb{R}^n$ , where  $m_i^{k|k}$  denotes the number of hyperplanes in the arrangement of the term *i*. The hyperplanes are said to be in a central arrangement if all hyperplanes pass through the origin, i.e  $A_i^{k|k}\nu = \mathbf{0}^{m_i^{k|k}}$  and a *general* arrangement otherwise. Therefore,  $\lambda_i^{k|k}(\nu) \in \mathbb{R}^{m_i^{k|k}}$  is the sign-vector function of the central hyperplane arrangement  $A_i^{k|k}$ , which indicates the half-space  $H_{il}^+$  or  $H_{il}^-$  a chosen  $\nu \in \mathbb{R}^n$  lies in, with respect to each of the hyperplanes in the arrangement.

The key observation now is that an arrangement of hyperplanes only produces a finite number of *cells*. A cell is formed by the intersection of the halfspaces of its hyperplane arrangement and is *uniquely* defined by its sign vector. Moreover, the sign vector is constant within the interior of a cell. Therefore, the sign-vector function  $\lambda_i^{k|k}(\nu)$  can only take on as many values as there are cells in the arrangement. Since  $y_{gi2}^{k|k}(\nu)$  is a function of the sign-vectors, it too can only take on as many values as there are cells in the hyperplane arrangement. Lastly, since  $g_i^{k|k}(\nu)$  of (2.5) is a function of the sign-vectors, it too can only take on as many values as cells in the arrangement.

The strategy to remove the backward recursive component of (2.5) follows from these observations. We can find the value of (2.5) for each cell by determining the set of sign-vectors of the hyperplane arrangement, since (2.5) only takes on as many values as there are cells. The idea is to then replace (2.5), completely, with a linear parameterization. This can be accomplished by solving a linear system of equations using the values of (2.5) and (a transformation of) the sign vectors. Sections 3.2 and 3.3 elaborate on this procedure.

# 3.2 Hyperplane Arrangements, Cells, and Cell Enumeration

Figure 3.1 depicts various two-dimensional central and general arrangements and their sign vectors. It is clear from Fig. 3.1 that each cell has a unique sign-vector, as crossing any hyperplane will flip the respective sign-vector value. A comparison of the general arrangements in Fig. 3.1 shows various geometrical configurations of the hyperplanes can also vary the number of cells. This can also occur in central arrangements as well.

For a central arrangement with m hyperplanes in n dimensions, the maximum number of cells the arrangement can have is given by

$$s_c(m,n) = 2\sum_{i=0}^{n-1} \binom{m-1}{i}.$$
(3.1)

Similarly, the maximum number of cells for general hyperplane arrangements can be written as

$$s_g(m,n) = \sum_{i=0}^n \binom{m}{i},\tag{3.2}$$



Figure 3.1: Cell enumeration examples of 4 hyperplanes in 2 dimensions for general (top right, 11 cells), central (left, 8 cells), and degenerate (bottom right, 9 cells) arrangements, depicting the hyperplane half-spaces (+/- signs) and cell sign-vectors (boxed vectors).

where  $s_c(m_i^{k|k}, n) \leq s_g(m_i^{k|k}, n)$  always holds [20]. If an arrangement (either central or general) is seen to have less than  $s_c(m_i^{k|k}, n)$  or  $s_g(m_i^{k|k}, n)$  cells, respectively, the arrangement is said to be degenerate. Therefore, a procedure is needed to find the set of all sign vectors of the hyperplane arrangement, as doing so will allow (2.5) to be evaluated in all cells. Furthermore, the procedure will need to find the sign vectors for any number of hyperplanes, in any dimension, and regardless of any potential degeneracy in the arrangement.

Such a technique is called a *cell-enumeration* algorithm for hyperplane arrangements, first proposed by [20]. Later, a more efficient 'incremental-enumeration' technique (coined by the authors as Inc-Enu) of [21] was proposed. The remainder of this section focuses on explaining how the challenging problem of cell enumeration (for central hyperplane arrangements) can be solved using Inc-Enu, and a high-level overview is given below<sup>1</sup>. If

<sup>&</sup>lt;sup>1</sup>The work of [14] proposes an even more computationally efficient version of the Inc-Enu algorithm, contributed by the author of this dissertation. The proposed Inc-Enu in [14] uses a GPU to conduct a parallel breadth-first search and a custom GPU simplex solver written in CUDA-C to solve the LPs. Moreover, it can solve multiple cell-enumeration problems simultaneously. The scheme is outlined in chapter 6 and relies heavily on details of this section.

the specifics of cell enumeration are not of interest to the reader, the remainder of this section can be skipped up to the last paragraph. For the interested reader, see below. See [21] for a more detailed discussion of Inc-Enu and [20] for an overview of the general 'reverse-search' class of methods.

For central arrangements, all sign vectors can be found by solving Phase-I feasibility linear programs  $(LPs)^2$  (see (3.3)). Phase-I LPs are used as it is not important *what* point in a cell is found; only that a point in space which satisfies the proposed sign-vector *can* be found. Naively, one could attempt  $2^m$  feasibility LPs to test all permutations of sign-vectors  $\{\pm 1\}^m$ , given an arrangement with *m*-hyperplanes. This strategy becomes highly intractable, however, as *m* grows larger than 10. The Inc-Enu algorithm, instead, solves a sequence of LPs, recursively building up (and verifying) sign vectors that belong to the arrangement from smaller partial sign sequences. If the LP is found infeasible, the partial sign sequence (or full sign-vector) attempted is now known to not belong to any given cell in the arrangement.

To initialize Inc-Enc, a 'root' point in space is arbitrarily initialized to be the cell located in the positive halfspace of all hyperplanes (so long as it does not fall perfectly on any hyperplane). If the (pseudo-randomly) declared root point is found to be in a negative halfspace of a particular hyperplane (which is common), the hyperplane coefficients are simply negated to flip which side defines the positive halfspace. At the end of the procedure, this negation can be undone by flipping the elements of the (solved for) sign-vectors at the indices corresponding to hyperplanes that were initially negated.

It should also be noted that for central arrangements, sign-vectors for all cells located in the positive halfspace of any chosen hyperplane are exact opposites of all the signvectors in the chosen hyperplane's negative halfspace (see Fig. 3.1 left). This implies we can simply choose an arbitrary hyperplane and look for all cells in its positive halfspace. Therefore, it is only required to solve for half of the cells in a central arrangement, and furthermore it is only required to enumerate over m-1 hyperplanes. We refer to the hyperplane whose *positive* halfspace is enumerated over as the 'symmetric generator'.

 $<sup>^{2}</sup>$ See [26] for details on the simplex and/or the primal-dual interior point algorithms, which can be used to solve LPs.

The symmetric generator can be taken to be the first hyperplane in the arrangement, for simplicity. Once all cells located in the positive halfspace of the symmetric generator are found, the sign-vectors can be negated and concatenated to the original set to determine the negative halfspace.

Figure 3.2 depicts an example of the Inc-Enu scheme for hyperplanes  $a_{\ell} \in \mathbb{R}^n, \ell \in [1, ..., m]$ , where m = 4 and dimension n = 3. Since the symmetric generator is fixed to be positive, the enumeration is carried out only on the other 3 hyperplanes. Therefore, a sign sequence for example of [-1, -1, -1] in Fig. 3.2 is truly [1, -1, -1, -1] after accounting for the symmetric generator. Also note that for the discussion below, the 'term' subscript i and 'estimation step' superscript k|k is discarded for brevity.

Inc-Enu is a recursive, depth-first-search algorithm. At each recursion level, it solves the Phase-I LP of the form

minimize 0 (3.3)  
s.t 
$$a_{\ell}^{T} \nu \geq 1$$
 if  $s_{\ell} = 1$   
 $a_{\ell}^{T} \nu \leq -1$  if  $s_{\ell} = -1$   
 $\ell \in J, \quad J \subset \{1, ..., m\}$ 

where  $s_{\ell} \in \{\pm 1\}$  is the sign-value being tested for hyperplane  $a_{\ell}$  and  $s \in \{\pm 1\}^{|J|}$  with  $|J| \leq m$  the cardinality of index set J. Note |J| also equals the recursion level. At each successive recursion level, an additional inequality constraint is added, growing the length of the tested sign sequence s by 1. The right-hand side of the inequality constraints is simply chosen as the partial sign sequence that is being tested for feasibility. This can be justified since cells of central hyperplane arrangements are unbounded and convex regions [26], which can also be seen in Fig. 3.1 (left). Therefore, if the (partial or full) sign sequence is valid, a point in space  $\nu$  can always be found that meets the constraint.

Initially, the algorithm already knows the sign-vector of the root-point (the all positive sign vector of dimension m-1), and recursively descends along the left branch (without any LP solve required) to the left-most leaf-node to store it. The algorithm then pops-



Figure 3.2: Incremental enumeration (Inc-Enu) tree for cell enumeration of a four hyperplane, three dimensional arrangement. Boxes in gray denote an LP was solved to validate the current sign sequence. Boxes in green denote no LP was needed to validate the sign sequence. Boxes in red denote the LP was infeasible and the proposed sign sequence is invalid. RL denotes the recursion level at which the sequence is formed.

up one level of recursion. At recursion level RL-2, there is nothing left to do, and the algorithm recurs back down again and proposes the sequence [1, 1, -1]. Here, it solves a Phase-I LP to check whether the new sequence is valid. If the solve is successful, the solution vector  $\nu$  now defines a new point in space. One can convince themselves that regardless of the current recursion level, after an LP is solved successfully, the LP solution  $\nu$  can immediately be used to construct a new and *complete* sign vector simply by checking  $\text{sgn}(\langle a_{|J|+\ell}, \nu \rangle), \forall l \in \{|J+1|, ..., m\}$  (i.e., for all remaining indices not present in J yet). In doing so, it recurs back down to recursion level RL-3 without the need of further LP solves. This is shown visually by the connected sequence of green left child boxes in Fig. 3.2. The process continues until the depth-first search recursively covers the full enumeration tree. In cell enumeration, it is common to refer to the LP solver as an *oracle* who answers *queries*.

If a given sign sequence is found infeasible (shown in red) by the oracle, Inc-Enu will pop up a recursion level instead of recuring down to depth m-1 to form the new signvector. We note that any sign sequences that are found infeasible by the oracle do not have left and right children and therefore are leaves of the tree. We see that 'left' child nodes (shown in green) of the parent are found without querying the oracle, whereas 'right' children (shown in red/black) denote where proposed sign sequences are assigned to the oracle for a query. We see that the enumeration in Figure 3.2 yields 7 feasible sign-vectors in the positive halfspace of the symmetric generator, implying there are 14 total cells in the (non-degenerate) arrangement. Note that the maximum number of cells is 14 for a four hyperplane, three-dimensional central arrangement as is given by (3.1). See chapter 6 for a pseduo-algorithm of Inc-Enu.

The Inc-Enu procedure, described above, can be applied to find the set of sign vectors for every central hyperplane arrangement  $A_i^{k|k}$ , for each term  $i \in [1, ..., N_t^{k|k}]$  (see footnote 1). The set of all sign-vectors of term i, i.e.,  $\lambda_i^{k|k}(\nu \in C_j) \in \{\pm 1\}^{m_i^{k|k}}, j \in$  $[1, ..., c_c(A_i^{k|k})]$ , stacked row-wise, can be written as  $B_i^{k|k} \in \{\pm 1\}^{c_c(A_i^{k|k}) \times m_i^{k|k}}$ , where  $C_j$ denotes a particular cell and  $c_c(A_i^{k|k})$  denotes the total number (count) of cells for the central hyperplane arrangement of term i, found by Inc-Enu. The matrix  $B_i^{k|k}$  is referred to as the *enumeration matrix* in the upcoming sections of this chapter.

### **3.3** A Basis Expansion for Sign-Vectors

Now that the set of sign-vectors  $\bar{\lambda}_{ij}^{k|k} = \lambda_i^{k|k}(\nu \in C_j)$ ,  $j \in [1, ..., c_c(A_i^{k|k})]$  have been found, equation (2.5) can be evaluated using  $\bar{\lambda}_{ij}^{k|k}$  for each cell  $C_j$ . Let  $\bar{g}_{ij}^{k|k} = g_i^{k|k}(\nu \in C_j) = g_i^{k|k}(y_{gi}(\nu \in C_j))$  denote the value of (2.5) evaluated in cell  $C_j$ . These values can be stored in a vector  $\bar{g}_i^{k|k} \in \mathbb{C}^{c_c(A_i^{k|k})}$ . The goal is now to parameterize all values in  $\bar{g}_i^{k|k}$  with a vector  $\alpha_i^{k|k}$ . The problem, is that

$$\bar{g}_i^{k|k} = B_i^{k|k} \alpha_i^{k|k} \tag{3.4}$$

would be an over-determined system of equations, as  $B_i^{k|k} \in \{\pm 1\}^{c_c(A_i^{k|k}) \times m_i^{k|k}}$  and  $m_i^{k|k} < c_c(A_i^{k|k})$ . Therefore, a least squares solution vector  $\alpha_i^{k|k}$  would produce non-zero residuals  $\bar{g}_i - B_i^{k|k} \alpha_i^{k|k}$ , which is not acceptable.

Instead, the enumeration matrix must be transformed in a way such that the resultant matrix has a rank of  $c_c(A_i^{k|k})$ . To do so, each sign-vector  $\bar{\lambda}_{ij}^{k|k}$  stored in the enumeration

matrix  $B_i^{k|k}$  can be transformed by a basis function  $S(\cdot)$ . The basis function chosen is a combinatorial expansion of the elements of the sign-vector. Specifically, the products of up to n and out of the  $m_i^{k|k}$  elements of each sign-vector. To illustrate the basis-vector expansion explicitly, consider a three-dimensional n = 3 three hyperplane m = 3 arrangement. A sign-vector could be expressed generally as  $\bar{\lambda} = [\sigma_1, \sigma_2, \sigma_3]^T, \sigma_k \in \{-1, 1\}$  with basis-vector  $S(\bar{\lambda}) = [1, \sigma_1, \sigma_2, \sigma_3, \sigma_1\sigma_2, \sigma_1\sigma_3, \sigma_2\sigma_3, \sigma_1\sigma_2\sigma_3]^T$ . The transformation takes combinations of the products up to three elements, as the dimension n = 3. The result is that  $S(\bar{\lambda}) \in \{\pm\}^{s_g(m,n)}$ , where  $s_g(m, n)$  is given by (3.2).

The combinatorial expansion of all sign-vectors in  $B_i^{k|k}$  is then grouped into a matrix  $\bar{S}_i^{k|k} \in \{\pm 1\}^{c_c(A_i^{k|k}) \times s_g(m_i^{k|k},n)}$  (*i.e.*, the basis matrix), where each basis-vector  $S(B_{i,j}^{k|k})$ ,  $\forall j = [1, ..., c_c(A_i^{k|k})]$  is stored row-wise in  $\bar{S}_i^{k|k}$ . Note that  $\bar{S}_i^{k|k}$  will be a square or wide matrix, as  $c_c(A_i^{k|k}) \leq s_c(m_i^{k|k}, n) \leq s_g(m_i^{k|k}, n)$  always holds true, and will also have a rank of  $c_c(A_i^{k|k})$  for any valid enumeration matrix. For further details on constructing these basis-vectors, see [14, 18, 23]. The system of equations

$$\bar{g}_i^{k|k} = \bar{S}_i^{k|k} \alpha_i^{k|k} \tag{3.5}$$

$$\alpha_i^{k|k} = \left(\bar{S}_i^{k|k}\right)^{\dagger} \bar{g}_i^{k|k} \tag{3.6}$$

seen in (3.5) can then be solved and  $\alpha_i^{k|k} \in \mathbb{C}^{s_g(m_i^{k|k},n)}$  of (3.6) is its solution. Since  $c_c(A_i^{k|k}) < s_g(m_i^{k|k},n)$  and  $\bar{S}_i^{k|k}$  is a full row-rank matrix, a particular solution of the under-determined linear problem in (3.5) can be obtained using a multitude of methods. Specifically, in [14] the least norm solution was adopted, where  $(\bar{S}_i^{k|k})^{\dagger}$  in (3.6) is the pseudo inverse of  $\bar{S}_i^{k|k}$ .

### 3.3.1 Parameterizing the G-Function

For any value of  $\nu \in \mathbb{R}^n$ , equation (2.5) is now equivalent to

$$g_{i}^{k|k}\left(\nu\right) = \frac{1}{2\pi} \left[ \frac{g_{r_{i}^{k|k}}^{k-1|k-1}\left(y_{gi1}^{k|k}(\nu) + h_{i}^{k|k}\right)}{jc_{i}^{k|k} + d_{i}^{k|k} + y_{gi2}^{k|k}(\nu)} - \frac{g_{r_{i}^{k|k}}^{k-1|k-1}\left(y_{gi1}^{k|k}(\nu) - h_{i}^{k|k}\right)}{jc_{i}^{k|k} - d_{i}^{k|k} + y_{gi2}^{k|k}(\nu)} \right] = S\left(\lambda_{i}^{k|k}(\nu)\right)^{T} \alpha_{i}^{k|k}$$

$$(3.7)$$

where  $S\left(\lambda_i^{k|k}(\nu)\right)$  simply expands the sign-vector of hyperplane arrangement  $A_i^{k|k}$  at estimation step k, for a chosen value of  $\nu$ . Note that  $y_{gi2}^{k|k} = \left(q_i^{k|k}\right)^T \lambda_i^{k|k}(\nu)$ . The question now becomes how  $\alpha_i^{k|k}$  can alternatively be used at the next estimation step k+1|k+1 to evaluate the backward-recursive numerator coefficients  $g_{r_i^{k+1|k+1}}^{k|k}\left(y_{gi1}^{k+1|k+1}(\nu) \pm h_i^{k+1|k+1}\right)$ . Finding this 'compressed' form of (2.5) would allow all similar terms to be combine together and all past terms from steps 1 to k of the characteristic function to be forgotten (excluding  $\alpha_i^{k|k}$ ). In section 3.4, the form of the compressed characteristic function is now formulated.

# 3.4 Equivalence of the Backward Recursive and Compressed Characteristic Functions

The general form of the compressed characteristic function can be uncovered by examining the linage of terms that the time-propagation and measurement updates steps create from the initial characteristic function's parameter set  $A^1, p^1, b^1$  (equation (2.3)). Explicitly, the measurement update at the estimation step 1|1, the time propagation step at 2|1, and the measurement update step at 2|2 will be considered and studied. These insights produce (3.19), which is the sought-after and compressed form of (2.5). The rules for time-propagating and measurement update for all estimation steps k > 2 are identical for the rules of steps 2|1 and 2|2, respectively. The insights presented in sections 3.1 to 3.3.1 will be considered as the transformations to the characteristic function are performed. Lastly, a numerical example is given in section 3.5 to explicitly show the equivalence of the forms. This example additionally serves as a good debugging tool for implementing the proposed estimator.

#### **3.4.1** Measurement Update at Estimation Step 1|1

Child terms are created each time a sensor measurement is processed by the measurement update step. At initialization (k = 1), there is only one term: the parameter set  $A^1, p^1, b^1$ , where  $A^1$  has n hyperplanes of dimension n. This 'parent' term will generate n new 'child' terms after the measurement update at step 1|1 and therefore the total number of terms will increase to n + 1. The additional term is simply the parent term itself. The general rules for the measurement update, derived in [16], are summarized below as

$$A_{tl}^{k|k} = \mu_{il}^{k|k-1} - \mu_{it}^{k|k-1} \in \mathbb{R}^{1 \times n}, \ \forall \ l = [1, ..., m_i^{k|k-1} + 1], \ l \neq t, \ A_t^{k|k} \in \mathbb{R}^{m_i^{k|k-1} \times n}$$
(3.8a)

$$\mu_{il}^{k|k-1} = \begin{cases} \frac{A_{il}^{k|k-1}}{\langle H_k, A_{il}^{k|k-1} \rangle} \in \mathbb{R}^{1 \times n}, & \text{if } l \le m_i^{k|k-1} \\ 0 \in \mathbb{R}^{1 \times n}, & \text{if } l = m_i^{k|k-1} + 1 \end{cases}, \quad A_{il}^{k|k-1} = A_i^{k|k-1}[l, :] \in \mathbb{R}^{1 \times n} \quad (3.8b)$$

$$p_{tl}^{k|k} = \begin{cases} p_{il}^{k|k-1} \operatorname{abs}(\langle H_k, A_{il}^{k|k-1} \rangle), & \text{if } l \le m_i^{k|k-1} \text{ and } l \ne t \\ \gamma, & \text{if } l = m_i^{k|k-1} + 1 \text{ and } l \ne t \end{cases}, \quad p_t \in \mathbb{R}^{m_i^{k|k-1}} \tag{3.8c}$$

$$b_t^{k|k} = \zeta_i \left( \mu_{it}^{k|k-1} \right)^T + b_i^{k|k-1} \quad \text{if } t \le m_i^{k|k-1} \quad \text{else } b_i^{k|k-1}, \quad b_t \in \mathbb{R}^n$$
(3.8d)

$$c_t^{k|k} = \zeta_i, \quad c_t^{k|k} \in \mathbb{R}$$
(3.8e)

$$d_t^{k|k} = \operatorname{abs}(\langle H_k, A_{it}^{k|k-1} \rangle) p_{it}^{k|k-1} \text{ if } t \le m_i^{k|k-1} \text{ else } \gamma, \quad d_t^{k|k} \in \mathbb{R}$$

$$(3.8f)$$

$$\zeta_i = z_2 - H_k b_i^{k|k-1}, \quad \zeta_i \in \mathbb{R}$$
(3.8g)

where the index variable  $t \in [1, ..., m_i^{k|k-1} + 1]$  specifies which child term of parent *i* is in question and  $\gamma \in \mathbb{R}_{++}$  is the Cauchy scaling parameter for the measurement noise of (2.1). Note,  $m_t^{k|k} = m_i^{k|k-1} = n$  for the first measurement update step.

Several properties of the measurement update step should be discussed. For the first measurement update at k = 1, the notation of the initial parameter set  $\{A^1, p^1, b^1\}$  is relaxed and equivalently referred to as  $\{A^{1|0}, p^{1|0}, b^{1|0}\}$ . For steps k > 1, there will be many parent terms  $i \in [1, ..., N_t^{k|k-1}]$  generating their own respective  $m_i^{k|k-1}+1$  child terms, where the total number of terms after a measurement update is given by (2.13). After the measurement update, all child terms (from all parents) are re-indexed continuously as  $i \in [1, ..., N_t^{k|k}]$ . Equation (3.8a) indicates all child terms t of parent i have  $m_i^{k|k-1}$  must  $m_i^{k|k-1}$  hyperplanes in their respective arrangements, although now these hyperplanes are different from those of the parent.

Equation (3.8a) reveals why term-coalignment is necessary, and a quick comment should be made. If  $\mu_{il}^{k|k-1} = \mu_{it}^{k|k-1}$  for  $l \neq t$  in (3.8a), the new hyperplane would be singular and (3.8b) would be infinite at k + 1. This is why the measurement coalignment step of section 2.1.1 is necessary, as it prevents this from occurring. If measurement coalignment is run,  $p_i^{k|k} \neq q_i^{k|k}$ , otherwise they are equal. We also see that  $\langle H_k, A_{il}^{k|k-1} \rangle$ must not equal zero in (3.8b). Picking a suitable  $A^1$  with respect to the dynamics  $\Phi_k$  and measurement model  $H_k$  resolves this concern. Also note that if  $H_k$  has more than one row, or there are multiple measurements from an array of sensors, the superscript k|kof all generated child parameters  $\{A_i^{k|k}, p_i^{k|k}, b_i^{k|k}\}, i \in [1, ..., N_t^{k|k}]$  are replaced with k|k-1and the measurement update routine above is simply re-run.

The index of the singularity 't = l' will be shown in this section to be extremely important (i.e when t = l during forming  $A_t^{k|k}$  of (3.8a) and  $p_t^{k|k}$ ). The singularity t = lis *excluded*, as seen in (3.8a). It causes an indexing issue when storing the child terms, as the index jumps from l - 1 to l + 1 to avoid t. Consequently, to deal with this jump in the index (and for continuity purposes), the parameters  $A_{tl}^{k|k} \in \mathbb{R}^{1 \times n}$ ,  $p_{tl}^{k|k} \in \mathbb{R}$  for l > tmust be shifted up by an index of 1 when stored for (3.8a) and (3.8c). For example, if t = 2 and l = 3, since l = t = 2 was excluded, the  $(\cdot)_{t=2,l=3}$  components are stored in computer memory at the l = 2 (not l=3) vector index. Therefore, it should be stressed that when any equation uses  $l \neq t$ , elements of l > t are truly stored then at l - 1.

The set of 'child' terms  $\{A_t^{1|1}, b_t^{1|1}, c_t^{1|1}, d_t^{1|1}, p_t^{1|1}\}, t \in [1, ..., n + 1]$  that the (parent) initial conditions generate is now constructed. Using these parameters, (2.8) and (2.9) can be called to evaluate the characteristic function for the conditional mean  $\hat{x}_1$  and covariance  $P_1$ . Measurement coalignment does not affect step k = 1 and is ignored for now, as all directions produced are independent if  $\operatorname{rank}(A^1) = n$ , which is a required condition. The coalignment step will have important ramifications at step 2|2, however, for the compressed form of the characteristic function.

Now we can apply the insights gained in sections 3.1 to 3.3.1 to the terms at 1|1. For each child parameter  $A_t^{1|1}$ , an enumeration matrix  $B_t^{1|1}$  is constructed using the Inc-Enu algorithm, which produces  $c_c(A_t^{1|1}) \leq s_c(m_t^{1|1}, n)$  sign-vectors. Next, the sign-vectors  $\bar{\lambda}_{t,j}^{1|1}$ ,  $j \in [1, ..., c_c(A_t^{1|1})]$  (that are stored in the rows of  $B_t^{1|1}$ ) are used to build  $\bar{S}_t^{1|1}$  through the basis expansion. Similarly,  $\bar{g}_t^{1|1}$  is constructed by evaluating (2.5) using the  $\bar{\lambda}_{t,j}^{1|1}$  as

$$\bar{g}_{t,j}^{1|1}(\bar{\lambda}_{i,j}^{1|1}) = \frac{1}{2\pi} \left[ \frac{1}{jc_t^{1|1} + d_t^{1|1} + (q_t^{1|1})^T \bar{\lambda}_{t,j}^{1|1}} - \frac{1}{jc_t^{1|1} - d_t^{1|1} + (q_t^{1|1})^T \bar{\lambda}_{t,j}^{1|1}} \right], \quad (3.9)$$

where the sign-vector  $\bar{\lambda}_{t,j}^{1|1}$  explicitly replaces the spectral variable  $\nu$  as the function argument of (3.9). This is an obvious change, as the sign-vector that  $\nu$  constructs has already been formed. Note that since we are at step 1|1, the numerators  $g_{r_t^{1|1}}^{0|0} \left(y_{gt1}^{1|1}(\nu) \pm h_t^{1|1}\right)$  of (2.5) are by definition equal to 1, as seen above. The parameter  $\alpha_t^{1|1}$  can now be constructed as  $\left(\bar{S}_t^{1|1}\right)^{\dagger} \bar{g}_t^{1|1}$  (i.e, equation (3.6)) for each child t, and will be used at 2|2 to evaluate the left and right-hand side numerators  $g_{r_t^{2|2}}^{1|1} \left(y_{gi1}^{2|2}(\nu) \pm h_t^{2|2}\right)$  of (2.5). Lastly, (although trivial for k = 1) the set of child terms t (of all parents i) are re-indexed continuously as  $i \in [1, ..., N_t^{k|k}]$  after completing the measurement update step. In general, this keeps the indexing consistent for generated child terms at k|k during time propagation at k+1|k.

### **3.4.2** Time Propagation at Estimation Step 2|1

The time propagation step to move from any estimation step k to k + 1 is straightforward. Only the parameters  $\{A_i^{k|k}, p_i^{k|k}, b_i^{k|k}\}$  generated by the measurement update step of each term *i* are updated. Below, we simply plug in k=1. This subroutine updates the parameter set  $\{A_i^{k|k}, b_i^{k|k}, p_i^{k|k}\}$  as

$$A_{i}^{k+1|k} = \begin{bmatrix} A_{i}^{k|k} \Phi_{k}^{T} \\ \Gamma_{k}^{T} \end{bmatrix}, \quad A_{i}^{k+1|k} \in \mathbb{R}^{(m_{i}^{k|k}+r) \times n} = \mathbb{R}^{m_{i}^{k+1|k} \times n}, \quad m_{i}^{k+1|k} = m_{i}^{k|k} + r \quad (3.10a)$$

$$b_i^{k+1|k} = \Phi_k b_i^{k|k}, \quad b_i^{k+1|k} \in \mathbb{R}^n$$
(3.10b)

$$p_i^{k+1|k} = \begin{bmatrix} p_i^{k|k} \\ \beta \end{bmatrix}, \quad p_i^{k+1|k} \in \mathbb{R}^{m_i^{k+1|k}} \quad \beta \in \mathbb{R}^r$$
(3.10c)

where  $\Phi_k \in \mathbb{R}^{n \times n}$  is the discrete time transition matrix of the underlying dynamic system and  $\Gamma_k \in \mathbb{R}^{n \times r}$  is the process noise matrix for the *r* independent Cauchy modeled process noises with vector pdf scaling parameter  $\beta$ . Note that  $\Gamma_k$  may coalign with the newly transformed hyperplanes  $A_i^{k|k} \Phi_k^T$ . Therefore, after running the time-propagation step as outlined in section 2.2, the value of  $m_i^{k+1|k}$  can only be given generally by  $m_i^{k|k} \leq m_i^{k+1|k} \leq m_i^{k|k} + r$ , which is the resultant hyperplane arrangement size of  $A_i^{k+1|k}$  and the number of elements in  $p_i^{k+1|k}$ .

Technically,  $q_i^{k+1|k}$  is just  $p_i^{k+1|k}$  of (3.10c), but has zeros in its portion of  $\beta$ . If elements of  $A_i^{k+1|k}$  are seen to coalign,  $p_i^{k+1|k}$  will reduce in size accordingly by the rules of (2.11) (by absorbing elements of  $\beta$  in its  $p_i^{k|k}$  component). Therefore, if coalignment is required, the parameter  $q_i^{k+1|k}$  will simply be the resultant  $p_i^{k+1|k}$  after coalignment, but with zeros located at the indices of the remaining  $\beta$  elements.

### 3.4.3 Measurement Update at Estimation Step 2|2

Once the new sensor measurement  $z_2$  is acquired, the time step index is incremented  $(k \leftarrow k+1)$  and now reads k=2|2. Moreover, the parameter set from time-propagation  $\{A_i^{2|1} \in \mathbb{R}^{m_i^{2|1} \times n}, b_i^{2|1} \in \mathbb{R}^n, p_i^{2|1} \in \mathbb{R}^{m_i^{2|1}}\}$  will now create its child terms. The indexing variable 't' is again used to specify the child terms, i.e, the t-th child term at 2|2 of the parent *i* from 2|1. Therefore, parent *i* creates children  $t \in [1, ..., m_i^{2|1} + 1]$ . Using (3.8), we can again write the set of child terms  $\{A_t^{2|2}, b_t^{2|2}, c_t^{2|2}, d_t^{2|2}, p_t^{2|2}\}$  that parent *i* generates.

Unlike step 1|1, the left and right-hand side numerator functions of (2.5) are no longer equal to 1. For k > 1, measurement update will also update past parent terms, as well as generate the new child terms at 2|2 via (3.8). In [16], equation (3.35) (re-presented below) shows how the argument to the left and right-hand side numerators of (2.5), i.e  $y_{gt1}^{2|2}(\nu)$ , is a function of both the updated parent parameters of k|k-1 and the new parameters at k|k, given as

$$y_{gt1}^{2|2}(\nu) = \sum_{\substack{l=1\\l\neq t}}^{m_i^{1|1}} \hat{\rho}_{il}^{2|1} \operatorname{sgn}(\langle A_{tl}^{2|2}, \nu \rangle), \quad l \in [1, ..., m_i^{1|1}]$$
(3.11a)

$$\hat{\rho}_{il}^{2|1} = q_{il}^{2|1} \hat{\lambda}_{il}^{2|1}, \quad \hat{\rho}_i^{2|1} \in \mathbb{R}^{m_i^{1|1}}$$
(3.11b)

$$\hat{\lambda}_{il}^{2|1}(\nu = H_k^T) = \operatorname{sgn}(\langle A_{il}^{2|1}, H_k^T \rangle), \quad l \in [1, ..., m_i^{1|1}], \quad \hat{\lambda}_i^{2|1} \in \mathbb{R}^{m_i^{1|1}},$$
(3.11c)

$$q_i^{2|1} = \begin{bmatrix} q_i^{1|1} \\ \mathbf{0}^r \end{bmatrix} \in \mathbb{R}^{m_i^{2|1}} = \mathbb{R}^{m_i^{1|1} + r},$$
(3.11d)

where  $\hat{\rho}_{il}^{2|1}$  is the updated parent parameter  $q_{il}^{2|1}$ . Careful attention to (3.11) is needed. First, the hat symbol on  $\hat{\rho}_{i}^{2|1}$ ,  $\hat{\lambda}_{i}^{2|1}$  of (3.11b) and (3.11c) indicates the additional r elements added due to time-propagation at 2|1 are unnecessary and are truncated. We do this to explicitly show that while  $y_{gt1}^{2|2}$  of (3.11a) does contain parameters from 2|1 and 2|2, it is only a summation over products between the original  $m_i^{1|1}$  parent elements  $q_i^{1|1}$  and functions of  $A_{il}^{2|1}$ ,  $A_{tl}^{2|2}$ ,  $l \in [1, ..., m_i^{1|1}]$ , which are the  $m_i^{1|1}$  transformed parent elements of 1|1 at 2|1 and 2|2 respectively. Therefore, (3.11) has been stated slightly differently than as it appeared in [16], namely, with the modified upper summation bound of (3.11a). Second,  $\hat{\lambda}_i^{2|1}$  of (3.11) is to be evaluated explicitly for  $\nu = H_k^T$ , as was shown in [16]. Note that (3.11d) is presented without consideration of coalignment in time propagation. If any hyperplanes of the parent are coaligned within time propagation, the **0**<sup>r</sup> of (3.11d) would need to be modified appropriately to reflect this.

We will see next that (3.11) is the key to forming the compressed characteristic function. Since  $y_{gt1}^{2|2}$  is passed to the backward recursive numerators as  $g_{r_i^{2|2}}^{1|1}(y_{gt1}^{2|2} \pm h_i^{2|2})$ , we see the form is

$$g_{r_{t}^{2|2}}^{1|1}\left(y_{gt1}^{2|2}(\nu)+h_{t}^{2|2}\right) = \frac{1}{2\pi} \left[\frac{1}{jc_{i}^{1|1}+d_{i}^{1|1}+h_{t}^{2|2}+y_{gt1}^{2|2}(\nu)} - \frac{1}{jc_{i}^{1|1}-d_{i}^{1|1}+h_{t}^{2|2}+y_{gt1}^{2|2}(\nu)}\right]$$
$$= \frac{1}{2\pi} \left[\frac{1}{jc_{i}^{1|1}+d_{i}^{1|1}+h_{t}^{2|2}+\sum_{\substack{l=1\\l\neq t}}^{m_{i}^{1|1}}\rho_{il}\mathrm{sgn}\left(\left(A_{tl}^{2|2}\right)^{T}\nu\right)} - \frac{1}{jc_{i}^{1|1}-d_{i}^{1|1}+h_{t}^{2|2}+\sum_{\substack{l=1\\l\neq t}}^{m_{i}^{1|1}}\rho_{il}\mathrm{sgn}\left(\left(A_{tl}^{2|2}\right)^{T}\nu\right)}\right], \quad (3.12)$$

and

$$g_{r_{t}^{2|2}}^{1|1} \left( y_{gt1}^{2|2}(v) - h_{t}^{2|2} \right) = \frac{1}{2\pi} \left[ \frac{1}{jc_{i}^{1|1} + d_{i}^{1|1} - h_{t}^{2|2} + y_{gt1}^{2|2}} - \frac{1}{jc_{i}^{1|1} - d_{i}^{1|1} - h_{t}^{2|2} + y_{gt1}^{2|2}} \right] \\ = \frac{1}{2\pi} \left[ \frac{1}{jc_{i}^{1|1} + d_{i}^{1|1} - h_{t}^{2|2} + \sum_{\substack{l=1\\l \neq t}}^{m_{i}^{1|1}} \rho_{il} \operatorname{sgn} \left( \left( A_{tl}^{2|2} \right)^{T} \nu \right)} - \frac{1}{jc_{i}^{1|1} - d_{i}^{1|1} - h_{t}^{2|2} + \sum_{\substack{l=1\\l \neq t}}^{m_{i}^{1|1}} \rho_{il} \operatorname{sgn} \left( \left( A_{tl}^{2|2} \right)^{T} \nu \right)} \right], \quad (3.13)$$

where

$$h_t^{2|2} = \begin{cases} \rho_{it}^{2|1}, & \text{if } t \le m_i^{1|1} \\ 0 & \text{if } t > m_i^{1|1}. \end{cases}$$
(3.14)

Equations (3.12) and (3.13) show explicitly that the g-function of step 1|1 is evaluated with the updated parameters at 2|2. Let us consider the case when our child term  $t \leq m_i^{1|1}$ . In (3.12) and (3.13), the effect of  $h_t^{2|2}$  is to simply define the term of 2|2 at 1|1 for the singularity index  $l = t \leq m_i^{1|1}$ , for the summation (to its right). Although the term  $A_t^{2|2}$  does not explicitly store the 'singular' hyperplane of index l = t, the  $\pm$  value in front of  $h_t^{2|2}$  in (3.12) and (3.13) indicates how the sign value of the singular hyperplane, i.e,  $\operatorname{sgn}(\langle \mathbf{0}^n, \nu \rangle) = \operatorname{sgn}(0)$  is defined for the left and right-hand side numerators. This definition of  $\operatorname{sgn}(0)$  was presented as part of the integration formula in [16], which says that  $\operatorname{sgn}(0)$  is defined as 1 for (3.12), and that  $\operatorname{sgn}(0)$  is defined as -1 for (3.13). This is where the  $\pm$  sign in front of  $h_t^{2|2}$  comes from. Note that for children  $t > m_i^{1|1}$ ,  $h_t^{2|2}$  has no effect on the summation (to its right) and is defined as 0. To make the observations that follow simpler, let us restate (3.12) and (3.13) using (3.11) and (3.14) for all children as

$$g_{r_{t}^{2|2}}^{1|1} \left( y_{gt1}^{2|2}(\nu) + h_{t}^{2|2} \right) = \frac{1}{2\pi} \left[ \frac{1}{jc_{i}^{1|1} + d_{i}^{1|1} + \left(q_{i}^{1|1}\right)^{T} \left(\hat{\lambda}_{i}^{2|1} \circ \hat{\lambda}_{t+}^{2|2}(\nu)\right)} - \frac{1}{jc_{i}^{1|1} - d_{i}^{1|1} + \left(q_{i}^{1|1}\right)^{T} \left(\hat{\lambda}_{i}^{2|1} \circ \hat{\lambda}_{t+}^{2|2}(\nu)\right)} \right]$$

$$(3.15a)$$

and

$$\begin{split} g_{r_t^{2|2}}^{1|1} \left( y_{gt1}^{2|2}(\nu) - h_t^{2|2} \right) \\ = & \frac{1}{2\pi} \left[ \frac{1}{jc_i^{1|1} + d_i^{1|1} + \left(q_i^{1|1}\right)^T \left(\hat{\lambda}_i^{2|1} \circ \hat{\lambda}_{t^-}^{2|2}(\nu)\right)} - \frac{1}{jc_i^{1|1} - d_i^{1|1} + \left(q_i^{1|1}\right)^T \left(\hat{\lambda}_i^{2|1} \circ \hat{\lambda}_{t^-}^{2|2}(\nu)\right)} \right] \\ (3.15b) \end{split}$$

where,

$$\hat{\lambda}_{t+l}^{2|2}(\nu) = \begin{cases} \operatorname{sgn}\left(\left(A_{tl}^{2|2}\right)^{T}\nu\right) & \text{if } l < t\\ \operatorname{sgn}\left(\left(A_{t(l-1)}^{2|2}\right)^{T}\nu\right) & \text{if } l > t , \quad l \in [1, ..., m_{i}^{1|1}], \quad \hat{\lambda}_{t+}^{2|2}(\nu) \in \{\pm 1\}^{m_{i}^{1|1}}\\ \operatorname{sgn}(0) = 1, & \text{if } l = t \end{cases}$$

$$(3.15c)$$

$$\hat{\lambda}_{t^{-l}}^{2|2}(\nu) = \begin{cases} \operatorname{sgn}\left(\left(A_{tl}^{2|2}\right)^{T}\nu\right) & \text{if } l < t\\ \operatorname{sgn}\left(\left(A_{t(l-1)}^{2|2}\right)^{T}\nu\right) & \text{if } l > t , \quad l \in [1, ..., m_{i}^{1|1}], \quad \hat{\lambda}_{t^{-}}^{2|2}(\nu) \in \{\pm 1\}^{m_{i}^{1|1}}\\ \operatorname{sgn}(0) = -1, & \text{if } l = t \end{cases}$$

$$(3.15d)$$

and the  $t^+$  and  $t^-$  subscripts on the vectors  $\hat{\lambda}_{t^+}^{2|2}(\nu)$ ,  $\hat{\lambda}_{t^-}^{2|2}(\nu)$  indicate how sgn(0) should be evaluated at the singularity index  $l = t \leq m_i^{1|1}$  for the left and right handside numerators (3.12) and (3.13). The symbol  $\circ$  above is the Hadamard product, which is element-wise vector multiplication. Note that for child  $t > m_i^{1|1}$ , we do not need to worry about the singularity index t in (3.15c) and (3.15d), as terms at step 1|1 only have  $m_i^{1|1}$  elements.

Using (3.15a) and (3.15b), the form of the compressed characteristic function can now be found. The restated equations of (3.15a) and (3.15b) are identical to that of (3.9), with the exception that  $\hat{\lambda}_i^{2|1}$  and  $\hat{\lambda}_{t^{\pm}}^{2|2}(\nu)$  now replace  $\bar{\lambda}_{i,j}$  of (3.9). Let us first examine the effect that  $\hat{\lambda}_i^{2|1}$  has. If we temporarily imagine that  $\hat{\lambda}_t^{2|2}(\nu)$  took on the values of the sign-vectors  $\bar{\lambda}_{i,j}^{1|1} \in \{\pm 1\}^{m_i^{1|1}}, \ j = [1, ..., c_c(A_i^{1|1})]$  of (3.9), equations (3.15a) and (3.15b) indicate the sign-vector  $\bar{\lambda}_{i,j}$  would be potentially flipped by values of  $\hat{\lambda}_i^{2|1}$ . As the sign-vectors  $\bar{\lambda}_{i,j}^{1|1}$  are the enumeration matrix  $B_i^{1|1}$ , it can be reasoned that the grouping  $\lambda_i^{1|1}(\nu) \circ \hat{\lambda}_i^{2|1}$  with the Hadamard product would effectively update  $B_i^{1|1}$  as

$$\hat{B}_{i}^{1|1} = \begin{bmatrix} \left( \bar{\lambda}_{i,1}^{1|1} \circ \hat{\lambda}_{i}^{2|1} \right)^{T} \\ \left( \bar{\lambda}_{i,1}^{1|1} \circ \hat{\lambda}_{i}^{2|1} \right)^{T} \\ \dots \\ \left( \bar{\lambda}_{i,1}^{1|1} \circ \hat{\lambda}_{i}^{2|1} \right)^{T} \end{bmatrix} = B_{i}^{1|1} \circ \hat{\lambda}_{i}^{2|1}, \qquad (3.16)$$

where the Hadamard product is relaxed to elementwise matrix vector multiplication, where the vector multiplies with each matrix row elementwise. Thus, the term  $\hat{\lambda}_i^{2|1}$  in (3.16) has really updated the parent  $\alpha_i^{1|1}$  as

$$\begin{aligned} \alpha_i^{2|1} &= \left( S(\hat{B}_i^{1|1}) \right)^{\dagger} \bar{g}_i^{1|1} = \left( S(B_i^{1|1} \circ \hat{\lambda}_i^{2|1}) \right)^{\dagger} \bar{g}_i^{1|1} \\ \alpha_i^{2|1} &= \left( S(B_i^{1|1}) \circ S(\hat{\lambda}_i^{2|1}) \right)^T \left( \left( S(B_i^{1|1}) \circ S(\hat{\lambda}_i^{2|1}) \right) \left( S(B_i^{1|1}) \circ S(\hat{\lambda}_i^{2|1}) \right)^T \right)^{-1} \bar{g}_i^{1|1} \\ \alpha_i^{2|1} &= S(\hat{\lambda}_i^{2|1}) \circ \left( S^{\dagger}(B_i^{1|1}) \bar{g}_i^{1|1} \right) = S(\hat{\lambda}_i^{2|1}) \circ \alpha_i^{1|1} \end{aligned}$$
(3.17)

and we refer to the updated parent alpha vector as  $\alpha_i^{2|1}$  since the parameters of 2|1 cause this modification to  $\alpha_i^{1|1}$ . Let us now examine the effect that  $\hat{\lambda}_{t^{\pm}}^{2|2}(\nu)$  has. Due to the t = l exclusion in measurement update (3.8), the elements  $\hat{\lambda}_{t+l}^{2|2}(\nu)$  and  $\hat{\lambda}_{t-l}^{2|2}(\nu)$  of (3.15c) and (3.15d) for children  $t \leq m_i^{1|1}$  at l = t contain sgn(0) and is defined to be  $\pm 1$ , respectively. As these two vectors are different from one another, the basis function  $S(\cdot)$  can therefore be applied to each  $\hat{\lambda}_{t+l}^{2|2}(\nu)$  and  $\hat{\lambda}_{t-l}^{2|2}(\nu)$ , which when multiplied by  $\alpha_i^{2|1}$  yields

$$S\left(\hat{\lambda}_{t^{+}}^{2|2}(\nu)\right)^{T}\alpha_{i}^{2|1} = g_{r_{t}^{2|2}}^{1|1}\left(y_{gt1}^{2|2}(\nu) + h_{t}^{2|2}\right), \qquad (3.18a)$$

$$S\left(\hat{\lambda}_{t^{-}}^{2|2}(\nu)\right)^{T}\alpha_{i}^{2|1} = g_{r_{t}^{2|2}}^{1|1}\left(y_{gt1}^{2|2}(\nu) - h_{t}^{2|2}\right),\tag{3.18b}$$

where (3.18) can now be used to remove the backward recursive component of (2.5) as

$$g_t^{2|2}(\nu) = \frac{1}{2\pi} \left[ \frac{S\left(\hat{\lambda}_{t^+}^{2|2}(\nu)\right)^T \alpha_i^{2|1}}{jc_t^{2|2} + d_t^{2|2} + d_t^{2|2} + \left(q_t^{2|2}\right)^T \lambda_t^{2|2}(\nu)} - \frac{S\left(\hat{\lambda}_{t^-}^{2|2}(\nu)\right)^T \alpha_i^{2|1}}{jc_t^{2|2} - d_t^{2|2} + \left(q_t^{2|2}\right)^T \lambda_t^{2|2}(\nu)} \right] \\ = \frac{1}{2\pi} \left[ \frac{S\left(\hat{\lambda}_{t^+}^{2|2}(\nu)\right)^T \alpha_i^{2|1}}{jc_t^{2|2} + d_t^{2|2} + y_{gt2}^{2|2}(\nu)} - \frac{S\left(\hat{\lambda}_{t^-}^{2|2}(\nu)\right)^T \alpha_i^{2|1}}{jc_t^{2|2} - d_t^{2|2} + y_{gt2}^{2|2}(\nu)} \right].$$
(3.19)

Equation (3.19) is the sought-after form of the compressed characteristic function. Note that  $\lambda_t^{2|2}(\nu)$  in the denominator of (3.19) is simply the sign-vector function of the hyperplane arrangement  $A_t^{2|2}$ , whereas the sign-vectors  $\hat{\lambda}_{t^+}^{2|2}(\nu)$  and  $\hat{\lambda}_{t^-}^{2|2}(\nu)$  require the specific manipulations of (3.15c) and (3.15d) to be formed. **Remark 1.** Some comments on the measurement coalignment step and equation (3.19) should now be mentioned. If two hyperplanes  $A_{tj}^{2|2}, A_{tl}^{2|2}$  where  $j < l \leq m_i^{1|1}$  coalign, the size of the vector  $\hat{\lambda}_{t\pm}^{2|2}(\nu)$  may be smaller then that of  $m_i^{1|1}$ . This is problematic because  $\alpha_i^{1|1}$  is of size  $s_g(m_i^{1|1}, n)$  and therefore  $S\left(\hat{\lambda}_{t\pm}^{2|2}(\nu)\right)$  will not match. It becomes crucial then that before evaluating  $S\left(\hat{\lambda}_{t\pm}^{2|2}(\nu)\right)$ , the sign-values corresponding to the coaligned hyperplanes must be reinserted into the vector  $\hat{\lambda}_{t\pm}^{2|2}(\nu)$  at the indices where they were removed from  $A_t^{k|k}$ . This can be done by re-growing  $\lambda_{t\pm}^{2|2}(\nu)$  by re-inserting the sign-value at indices j into l. It should also be noted that if the two hyperplane normals that coalign (i.e.,  $A_{tj}^{2|2}, A_{tl}^{2|2}$ ) point in opposite directions, the reinserted sign-value will need to be flipped. Additionally, coalignment can happen at more than two indices and for different pairings of indices simultaneously. Viewing the pseudo algorithms given in chapter 6 and the source code itself can help bring more clarity to these points.

The conditional mean  $\hat{x}_2$  and covariance  $P_2$  can now be generated at 2|2 using (2.8) and (2.9) by substituting in the compressed form of the g-function (3.19). Forming  $\alpha_t^{2|2}$  at step 2|2 is very similar to step 1|1. Inc-Enu is first run to generate the enumeration matrix  $B_t^{2|2} \in \{\pm 1\}^{c_c(A_t^{2|2}) \times m_t^{2|2}}$  using  $A_t^{2|2}$ . Then, the basis matrix  $\bar{S}_t^{2|2} \in \{\pm 1\}^{c_c(A_t^{2|2}) \times s_g(m_t^{2|2},n)}$  is formed by using the basis function expansion on the rows of  $B_t^{2|2}$ . Next,  $\bar{g}_t^{2|2} \in \mathbb{C}^{c_c(A_t^{2|2}) \times s_g(m_t^{2|2},n)}$  is formed by evaluating (3.19) using the rows of  $B_t^{2|2}$ . Lastly,  $\alpha_t^{2|2} = \left(\bar{S}_t^{2|2}\right)^{\dagger} \bar{g}_t^{2|2} \in \mathbb{C}^{s_g(m_t^{2|2},n)}$ is constructed by solving the resulting system of equations.

All of the parameter generation for the compressed characteristic function for step 2|2 is now outlined. The procedure to compress the characteristic function to any step k|kfollows immediately by repeating the step 2|2 procedure in its entirety. There exists one small indexing caveat. The presented form of (3.19) uses the child indexing variable t, making it clear that child t was generated from parent i. However, since there are many children  $t \in [1, ..., m_i^{k|k} + 1]$  from many different parents i, the index t is not unique. To write (3.19) generally for all children generated at any measurement update step k|k, we reindex all children at step k|k as  $i \in [1,...,N_t^{k|k}]$  and write

$$g_{i}^{k|k}(\nu) = \frac{1}{2\pi} \left[ \frac{S_{\psi_{i}}^{+} \left(\hat{\lambda}_{i}^{k|k}(\nu)\right)^{T} \alpha_{i}^{k|k-1}}{jc_{i}^{k|k} + d_{i}^{k|k} + \left(q_{i}^{k|k}\right)^{T} \lambda_{i}^{k|k}(\nu)} - \frac{S_{\psi_{i}}^{-} \left(\hat{\lambda}_{i}^{k|k}(\nu)\right)^{T} \alpha_{i}^{k|k-1}}{jc_{i}^{k|k} - d_{i}^{k|k} + \left(q_{i}^{k|k}\right)^{T} \lambda_{i}^{k|k}(\nu)} \right] \\ = \frac{1}{2\pi} \left[ \frac{S_{\psi_{i}}^{+} \left(\hat{\lambda}_{i}^{k|k}(\nu)\right)^{T} \alpha_{i}^{k|k-1}}{jc_{i}^{k|k} + d_{i}^{k|k} + y_{gi}^{k|k}(\nu)} - \frac{S_{\psi_{i}}^{-} \left(\hat{\lambda}_{i}^{k|k}(\nu)\right)^{T} \alpha_{i}^{k|k-1}}{jc_{i}^{k|k} - d_{i}^{k|k} + y_{gi}^{k|k}(\nu)} \right].$$
(3.20)

where for the general step k,

$$g_{i}^{k|k}(\nu) \in \mathbb{C}, \quad g_{i}^{k|k}(\nu) \in \bar{g}_{i}^{k|k}, \quad \bar{g}_{i}^{k|k} \in \mathbb{C}^{c_{c}(A_{i}^{k|k})},$$
$$y_{g_{i}}^{k|k} = \left(q_{i}^{k|k}\right)^{T} \lambda_{i}^{k|k}(\nu) \in \mathbb{R}, \quad q_{i}^{k|k} \in \mathbb{R}^{m_{i}^{k|k}}, c_{i}^{k|k}, d_{i}^{k|k} \in \mathbb{R}.$$
(3.21)

$$\lambda_{il}^{k|k}(\nu) = \text{sgn}\left(\langle A_{il}^{k|k}, \nu \rangle\right), \quad l \in [1, ..., m_i^{k|k}], \quad \lambda_i^{k|k}(\nu) \in \{\pm 1\}^{m_i^{k|k}}, \tag{3.22a}$$

$$\hat{\lambda}_{i}^{k|k}(\nu) = \lambda_{i}^{k|k}(\nu)[1:m_{i}^{k|k}-r], \quad \hat{\lambda}_{i}^{k|k}(\nu) \in \{\pm 1\}^{m_{i}^{k|k}-r}$$
(3.22b)

$$\hat{\lambda}_{il}^{k|k-1}(\nu = H_k^T) = \lambda_{il}^{k|k}(H_k^T), \quad l \in [1, ..., m_i^{k|k} - r], \quad \hat{\lambda}_{il}^{k|k-1} \in \{\pm 1\}^{m_i^{k|k} - r}$$
(3.22c)

and

$$\alpha_i^{k|k-1} = S\left(\hat{\lambda}_i^{k|k-1}(\nu = H_k^T)\right) \circ \alpha_i^{k-1|k-1} = \operatorname{diag}\left(S\left(\hat{\lambda}_i^{k|k-1}(H_k^T)\right)\right) \alpha_i^{k-1|k-1}.$$
 (3.23)

Some slight syntactical sugar has been added to (3.19) on the basis function  $S(\cdot)$ , but nevertheless, (3.19) and (3.20) are identical in meaning. Note that in this representation, the argument to  $S(\cdot)$  is now simply the (clipped) sign-vector  $\hat{\lambda}_i^{k|k}(\nu)$  of (3.22b), which is the first  $m_i^{k-1|k-1} = m_i^{k|k} - r$  components of the general sign-vector function  $\lambda_i^{k|k}(\nu)$  (keep in mind remark 1 above). The variable  $\psi_i = t$  stores the index of the singularity of the generated child  $i \in [1, ..., N_t^{k|k}]$ , where i is used for continuous indexing of all children now. The notation  $S_{\psi_i}^{\pm}(\cdot)$  is a useful syntax that denotes the procedure to first form  $\hat{\lambda}_{t^{\pm}}^{k|k}$ of (3.15c) and (3.15d) from the general sign-vector  $\hat{\lambda}_i^{k|k}(\nu)$  function and to then expand it by the basis function  $S(\cdot)$ . The subscript  $(\cdot)_{\psi_i}$  informs the basis function that either a  $\pm 1$  should be first inserted at the  $\psi_i$ -th index in the sign-vector  $\hat{\lambda}(\nu)$ . The superscript  $(\cdot)^{\pm}$  informs the basis function whether a  $\pm 1$ , respectively, should then be inserted at the index  $\psi_i$ . Note that since the backward recursive numerator has been removed,  $y_{gi1}^{k|k}(\nu)$ is no longer needed, and  $y_{gi2}^{k|k}(\nu)$  is written simply as  $y_{gi}^{k|k}(\nu)$  in (3.20).  $\alpha_i^{k|k-1}$  is still the updated parent parameter vector of child *i*.

This section showed explicitly how to form the compress characteristic function, and how its formulation was discovered. It should be mentioned that as it stands, (3.20) has not been able to be formed directly through re-deriving the integration formula of [16]. The sign-vector basis expansion of  $S(\cdot)$  has made re-deriving this already challenging integral even more so. However, by examining the components of (2.5), as was done in this section, [14] first showed the formulation of (3.20) is equivalent. Moreover, there is an ample amount of numerical evidence to suggest this equivalence is certainly true. The example presented in the next section shows the numerical equivalence of (2.5) and (3.20) using a three-state example that is presented in section 8.1. The upcoming example can be skipped if the reader does not need more convincing. The example provided, however, does serve as a good benchmark for one's understanding of the forms presented thus far.

### **3.5** Numerical Example of Equivalence

Here we walk through a numerical example of the old and new characteristic functions, and show they are indeed equivalent. This problem is taken from section 8.1, used in both [14, 16]. The dynamic system with scalar process and measurement noises in question is:

$$\Phi = \begin{bmatrix} 1.40 & -0.60 & -1.00 \\ -0.20 & 1.00 & 0.50 \\ 0.60 & -0.60 & -0.20 \end{bmatrix}, \quad \Gamma = \begin{bmatrix} 0.10 \\ 0.30 \\ -0.20 \end{bmatrix}, \quad H = \begin{bmatrix} 1.00 \\ 0.50 \\ 0.20 \end{bmatrix}^{T}, \quad \beta = 0.1 \quad \gamma = 0.2$$
(3.24)

where  $\{\Phi, \Gamma, H, \beta, \gamma\}$  is the transition matrix, the control matrix, the measurement matrix, process noise scale parameter and measurement noise scale parameter, respectively. The number of states is n = 3, and process noises is r = 1. The characteristic function processes an initial measurement at step k = 1 using the initial conditions provided of

$$A^{1|0} = \begin{bmatrix} 1.0 & 0.0 & 0.0\\ 0.0 & 1.0 & 0.0\\ 0.0 & 0.0 & 1.0 \end{bmatrix}, \quad p^{1|0} = \begin{bmatrix} 0.10\\ 0.08\\ 0.05 \end{bmatrix}, \quad b^{1|0} = \begin{bmatrix} 0.0\\ 0.0\\ 0.0 \end{bmatrix}, \quad z_1 = 0.0567$$
(3.25)

We evaluate one of the 4 sets of (parent) terms at k = 1|1 and one of the 5 sets of (child) terms this parent term will generate at 2|2. We use the notation set-up in section 3.4 and choose the 'parent' as i = 1 at 1|1 and its 'child' t = 2 at 2|2 for this example. Meaning, we evaluate the first of the four terms i = 1 at step 1|1, and then evaluate the second of the five children that i = 1 will generate, i.e t = 2 at step 2|2.

### Construction of the parameterization $\alpha_{i=1}^{1|1}$ for term i = 1 at step 1|1

Using the the measurement update (3.8) with the first measurement  $z_1 = 0.056659$ , the initial conditions generate n + 1 = 4 sets of terms  $\{A_i^{1|1} \in \mathbb{R}^{m_i^{1|1} \times n}, p_i^{1|1} \in \mathbb{R}^{m_i^{1|1}}, q_i^{1|1} \in \mathbb{R}^{m_i^{1|1}}, b_i^{1|1} \in \mathbb{R}^n, c_i^{1|1} \in \mathbb{R}, d_i^{1|1} \in \mathbb{R}\}$  where i = [1, ..., 4] indexes these sets and  $m_i^{1||1} = n = 3$ . For i = 1, our term set for step 1|1 is

$$A_{i=1}^{1|1} = \begin{bmatrix} \frac{A_2^{1|0}}{\langle HA_1^{1|0} \rangle} - \frac{A_1^{1|0}}{\langle HA_1^{1|0} \rangle} \\ \frac{A_3^{1|0}}{\langle HA_3^{1|0} \rangle} - \frac{A_1^{1|0}}{\langle HA_1^{1|0} \rangle} \\ - \frac{A_1^{1|0}}{\langle HA_1^{1|0} \rangle} \end{bmatrix} = \begin{bmatrix} -1.0 & 2.0 & 0.0 \\ -1.0 & 0.0 & 5.0 \\ -1.0 & 0.0 & 0.0 \end{bmatrix}, \quad A_l^{1|0} = A^{1|0}[l, :] \in \mathbb{R}^{1 \times 3} \quad l \in [1, 2, 3]$$

$$(3.26a)$$

$$p_{i=1}^{1|1} = \begin{bmatrix} p_2^{1|0} \operatorname{abs}(\langle H, A_2^{1|0} \rangle) \\ p_3^{2|1} \operatorname{abs}(\langle H, A_3^{1|0} \rangle) \\ \gamma \end{bmatrix} = \begin{bmatrix} 0.04 \\ 0.01 \\ 0.20 \end{bmatrix}, \quad p_l^{1|0} = p^{1|0}[l] \in \mathbb{R}, \quad p_{i=1}^{1|1} = q_{i=1}^{1|1}$$
(3.26b)

$$b_{i=1}^{1|1} = (z_1 - Hb_i^{2|1}) \left(\frac{A_1^{1|0}}{\langle H, A_1^{1|0} \rangle}\right)^T + b^{1|0} = \begin{bmatrix} 0.056659\\0.0\\0.0\\0.0 \end{bmatrix}$$
(3.26c)

$$c_{i=1}^{1|1} = (z_1 - Hb^{1|0}) = 0.056659$$

$$(3.26d)$$

$$d^{1|1} = abc(/H - A^{1|0})a^{1|0} = 0.1$$

$$(3.26c)$$

$$d_{i=1}^{1|1} = \operatorname{abs}(\langle H, A_1^{1|0} \rangle) p_1^{1|0} = 0.1 \tag{3.26e}$$

Now we can generate  $\alpha_{i=1}^{1|1}$  using (3.6) by forming  $B_{i=1}^{1|1}, \bar{S}_{i=1}^{1|1}, \bar{g}_{i=1}^{1|1}$ . The (evaluated) sign-vectors  $\bar{\lambda}_{1,j}$  are found by the Inc-Enu algorithm and then grouped row-wise as

where  $B_{i=1}^{1|1}$  is the enumeration matrix. We see the hyperplane arrangement  $A_{i=1}^{1|1}$  has  $c_c(A_{i=1}^{1|1}) = 8$  cells  $C_j$ ,  $j \in [1, ..., 8]$  where the sign-vectors  $\lambda_{i=1}^{1|1}(\nu \in C_j) = \bar{\lambda}_{i=1,j}^{1|1}$  locate all cells of  $A_{i=1}^{1|1}$ .

Note the ordering of the rows does not matter. Now,  $\bar{g}_{i=1}^{1|1}$  is evaluated using  $c_{i=1}^{1|1}, d_{i=1}^{1|1}, q_{i=1}^{1|1}$  of (3.26b), (3.26d) and (3.26e) using each  $\bar{\lambda}_{i=1,j}$  (i.e, in every cell  $\nu \in C_j$ ) of (3.27) as

$$\bar{g}_{i=1} = \begin{bmatrix} g_{i=1}(\nu \in C_1) \\ g_{i=1}(\nu \in C_2) \\ \vdots \\ g_{i=1}(\nu \in C_7) \\ g_{i=1}(\nu \in C_8) \end{bmatrix} = \begin{bmatrix} -0.809 - j0.993 \\ -0.809 + j0.993 \\ -0.485 + j0.279 \\ -0.485 - j0.279 \\ -0.560 + j0.367 \\ -0.560 - j0.367 \\ -0.560 - j0.367 \\ -0.788 - j1.440 \\ -0.788 + j1.440 \end{bmatrix} \in \mathbb{C}^{s_c(m_{i=1}^{1|1}, n)} = \mathbb{C}^{s_c(3,3)} = \mathbb{C}^8.$$

The basis-matrix  $S(B_{i=1}^{1|1})$  is constructed from the row-wise expansion of the enumeration matrix  $B_{i=1}^{1|1}$  and is given as

Now, we can use the relation (3.6) and find the parameterization of  $\bar{g}_{i=1}^{1|1}$  for all  $\nu \in \mathbb{R}^3$ ) as

$$\alpha_{i=1}^{1|1} = S^{\dagger}(B_{i=1}^{1|1})\bar{g}_{i=1} = \begin{bmatrix} -0.660741 + 0.0j \\ 0.0 - 0.447094j \\ 0.0 - 0.134383j \\ 0.0 + 0.770581j \\ 0.023981 + 0.0j \\ 0.137843 + 0.0j \\ 0.013481 + 0.0j \\ 0.0 + 0.089900j \end{bmatrix} \in \mathbb{C}^{s_g(m_{i=1}^{1|1}, n)} = \mathbb{C}^{s_g(3,3)} = \mathbb{C}^8.$$

Now,  $\alpha_{i=1}^{1|1}$  will be used to construct the numerator coefficients of (2.5) at step 2|2 below.

#### Generating child term t = 2 at step 2|2 of parent term i = 1 at step 1|1

The time step now moves from 1|1 to 2|1 and the parameters  $\{A_i^{1|1} \in \mathbb{R}^{m_i^{1|1} \times n}, p_i^{1|1} \in \mathbb{R}^{m_i^{1|1}}, b_i^{1|1} \in \mathbb{R}^n\}$  are now updated by the time propagation routine. Using (3.10), the parameter set for i = 1 at step 2|1 becomes

$$\begin{split} A_{i=1}^{2|1} &= \begin{bmatrix} A_{i=1}^{1|1} \Phi^T \\ \Gamma^T \end{bmatrix} = \begin{bmatrix} -2.60 & 2.20 & -1.80 \\ -6.40 & 2.70 & -1.60 \\ -1.40 & 0.20 & -0.60 \\ 0.10 & 0.30 & -0.20 \end{bmatrix} \\ A_i^{2|1} &\in \mathbb{R}^{(m_i^{1|1}+r) \times n} = \mathbb{R}^{m_i^{2|1} \times n}, \quad m_i^{2|1} = m_i^{1|1} + r = 3 + 1 = 4 \\ b_i^{2|1} &= \Phi b_i^{1|1} = \begin{bmatrix} 0.079322 \\ -0.011332 \\ 0.033995 \end{bmatrix}, \qquad b_i^{2|1} \in \mathbb{R}^n \\ p_i^{2|1} &= \begin{bmatrix} p_i^{1|1} \\ \beta \end{bmatrix} = \begin{bmatrix} 0.04 \\ 0.01 \\ 0.20 \\ 0.10 \end{bmatrix}, \qquad p_i^{2|1} \in \mathbb{R}^{m_i^{2|1}} \quad \beta \in \mathbb{R}^r. \end{split}$$

Next, the time step moves from 2|1 to 2|2 and our term set of i = 1 that we have followed will now generate  $m_{i=1}^{2|1} + 1 = 5$  child terms at 2|2. For our example, we will choose to follow the second child term t = 2 (of the five) at 2|2 that (our now parent) i = 1 will generate. The term set for child t = 2 (generated by measurement update (3.8)) given the second measurement  $z_2 = -0.14275$  is

$$\begin{split} A_{t=2}^{2|2} &= \begin{bmatrix} \frac{A_{i=1,1}^{2|1}}{\langle H, A_{i=1,1}^{2|1} \rangle} - \frac{A_{i=1,t=2}^{2|1}}{\langle H, A_{i=1,t=2}^{2|1} \rangle} \\ \frac{A_{i=1,3}^{2|1}}{\langle H, A_{i=1,4}^{2|1} \rangle} - \frac{A_{i=1,t=2}^{2|1}}{\langle H, A_{i=1,t=2}^{2|1} \rangle} \\ \frac{A_{i=1,4}^{2|1}}{\langle H, A_{i=1,4}^{2|1} \rangle} - \frac{A_{i=1,t=2}^{2|1}}{\langle H, A_{i=1,t=2}^{2|1} \rangle} \\ - \frac{A_{i=1,t=2}^{2|1}}{\langle H, A_{i=1,t=2}^{2|1} \rangle} \end{bmatrix} \\ &= \begin{bmatrix} 0.206043 & -0.680002 & 0.669790 \\ -0.205891 & 0.361948 & 0.124584 \\ -0.715616 & 1.931365 & -1.250333 \\ -1.191806 & 0.502793 & -0.297952 \end{bmatrix}, \quad (3.29a) \\ &A_{i=1,j}^{2|1} = A_{i=1}^{2|1}[j,:] \in \mathbb{R}^{1\times n}, \quad j \in [1, ..., m_t^{2|2}], \quad m_{t=2}^{2|2} = m_{i=1}^{2|1} = 4 \\ p_{i=1,1}^{2|1} \operatorname{abs}(\langle H, A_{i=1,1}^{2|1} \rangle) \\ &p_{t=2}^{2|2} = \begin{bmatrix} p_{i=1,1}^{2|1} \operatorname{abs}(\langle H, A_{i=1,1}^{2|1} \rangle) \\ p_{i=1,3}^{2|1} \operatorname{abs}(\langle H, A_{i=1,3}^{2|1} \rangle) \\ p_{i=1,4}^{2|1} \operatorname{abs}(\langle H, A_{i=1,4}^{2|1} \rangle) \\ &\gamma \end{bmatrix} \\ &= \begin{bmatrix} 0.0744 \\ 0.021 \\ 0.20 \end{bmatrix}, \quad p_{i=1,j}^{2|1} = p_{i=1}^{2|1}[j] \in \mathbb{R} \quad j \in [1, ..., 4] \\ (3.29c) \end{bmatrix} \end{aligned}$$

$$q_{t=2}^{2|2} = p_{t=2}^{2|2} \tag{3.29d}$$

$$b_{t=2}^{2|2} = \left(z_2 - Hb_{i=1}^{2|1}\right) \left(\frac{A_{i=1,t=2}^{2|1}}{\langle H, A_{i=1,t=2}^{2|1} \rangle}\right)^T + b_{i=1}^{2|1} = \begin{bmatrix} -0.186700\\ 0.100896\\ -0.032510 \end{bmatrix}$$
(3.29e)

$$c_{t=2}^{2|2} = (z_2 - Hb_{i=1}^{2|1}) = -0.223209$$
(3.29f)

$$d_{t=2}^{2|2} = \operatorname{abs}(\langle H, A_{i=1,t=2}^{2|1} \rangle) p_{i=1,t=2}^{2|1} = 0.053700.$$
(3.29g)

Note how the child hyperplane arrangement  $A_{t=2}^{2|2}$  of (3.29a) avoids creating a singular hyperplane at  $A_{t=2,l=2}^{2|2}$  by shifting all l > t = 2 up as mentioned in section 3.4. Equation (3.29) gives the necessary parameters to evaluate the non-recursive G-function (3.20). However, the recursive G-function (2.5) is also a function of updated parent terms, i.e,  $y_{gt1}^{2|2}$ , which is affected by the measurement update step by evaluating (3.11) (and choosing

any arbitrary value for  $\nu$ ) as

$$\nu = \begin{bmatrix} 1.0, 1.0, 1.0 \end{bmatrix}^T \in \mathbb{R}^n$$

$$y_{g(t=2)1}^{2|2} = \sum_{\substack{l=1\\l\neq(t=2)}}^{m_{i=1}^{1|1}} \hat{\rho}_{i=1,l}^{2|1} \operatorname{sgn}(\langle A_{t=2,l}^{2|2}, \nu \rangle) = \sum_{\substack{l=1\\l\neq(t=2)}}^{m_{i=1}^{1|1}} \hat{\rho}_{i=1,l}^{2|1} \hat{\lambda}_{t=2,l}^{2|2}(\nu) = \begin{bmatrix} -0.04\\ -0.20 \end{bmatrix}^T \begin{bmatrix} 1\\ 1 \end{bmatrix} = -0.24,$$
(3.30b)

$$\hat{\lambda}_{t^{\pm}=2}^{2|2}(\nu) = \left[ \operatorname{sgn}(\langle A_{t=2,1}^{2|2}, \nu \rangle) \quad \operatorname{sgn}(0) \quad \operatorname{sgn}(\langle A_{t=2,2}^{2|2}, \nu \rangle) \right]^{T} = \begin{bmatrix} 1 & \operatorname{sgn}(0) & 1 \end{bmatrix}^{T}$$
(3.30c)

$$\hat{\rho}_{i=1}^{2|1} = \begin{bmatrix} q_{i=1,1}^{-1} \hat{\lambda}_{i=1,1}^{-1} \\ q_{i=1,2}^{2|1} \hat{\lambda}_{i=1,t=2}^{2|1} \\ q_{i=1,3}^{2|1} \hat{\lambda}_{i=1,3}^{2|1} \end{bmatrix} = \begin{bmatrix} -0.04 \\ -0.01 \\ -0.20 \end{bmatrix}$$
(3.30d)

$$\hat{\lambda}_{i=1}^{2|1} = \begin{bmatrix} \operatorname{sgn}(\langle H, A_{i=1,1}^{2|1} \rangle) \\ \operatorname{sgn}(\langle H, A_{i=1,t=2}^{2|1} \rangle) \\ \operatorname{sgn}(\langle H, A_{i=1,3}^{2|1} \rangle) \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$
(3.30e)

$$q_{i=1}^{2|1} = \begin{bmatrix} q_{i=1}^{1|1} \\ \mathbf{0}^r \end{bmatrix} = \begin{bmatrix} 0.04 & 0.01 & 0.2 & 0 \end{bmatrix}^T$$
(3.30f)

$$h_{t=2}^{2|2} = \hat{\rho}_{i=1,t=2} = -0.01 \tag{3.30g}$$

where we have picked an arbitrary  $\nu \in \mathbb{R}^3$  and most importantly: since the  $l \neq t$  singularity of (3.30c) within  $A_{t=2}^{2|2}$  has already been removed by (3.29a) it must be dealt with carefully; here, by remembering that a singular hyperplane would have been created at index l = 2 in  $A_{t=2,l=t=2}^{2|2}$  if not first removed by (3.29a) and inserting a sgn(0) in (3.30c) at this position. Lastly, before we show the equivalence of the G's, let us finish here by remembering that

$$\lambda_{t=2,l}^{2|2}(\nu) = \operatorname{sgn}(\langle A_{t=2,l}^{2|2}, \nu \rangle), \quad l \in [1, ..., m_{t=2}^{2|2}], \quad \lambda_{t=2}^{2|2}(\nu) \in \mathbb{R}^{m_{t=2}^{2|2}}$$
(3.31)

$$y_{g(t=2)1} = \left(q_{t=2}^{2|2}\right)^T \lambda_{t=2}^{2|2}(\nu), \quad y_{g(t=2)1} \in \mathbb{R}$$
(3.32)

and for our example, using our declared  $\nu = [1.0, 1.0, 1.0]^T$  vector, we have

$$\lambda_{t=2,l}^{2|2}(\nu) = \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix}^T$$
$$y_{g(t=2)1} = 0.1374$$

#### Equivalence of G-functions (2.5) and (3.20) for child term t = 2

Now, finally, we are ready to evaluate (2.5) and (3.20). We start with (2.5). Its left and right hand side numerators are

$$g_{r_{t=2}^{2|2}}^{1|1} \left( y_{g(t=2)1}^{2|2}(\nu) + h_{t=2}^{2|2} \right) = \frac{1}{2\pi} \left[ \frac{1}{jc_{i=1}^{1|1} + d_{i=1}^{1|1} + h_{t=2}^{2|2} + y_{g(t=2)1}^{2|2}} - \frac{1}{jc_{i}^{1|1} - d_{i}^{1|1} + h_{t}^{2|2} + y_{g(t=2)1}^{2|2}} \right]$$

$$g_{r_{t=2}^{2|2}}^{1|1} \left( y_{g(t=2)1}^{2|2}(\nu) - h_{t=2}^{2|2} \right) = \frac{1}{2\pi} \left[ \frac{1}{jc_{i=1}^{1|1} + d_{i=1}^{1|1} - h_{t=2}^{2|2} + y_{g(t=2)1}^{2|2}} - \frac{1}{jc_{i=1}^{1|1} - d_{i=1}^{1|1} - h_{t=2}^{2|2} + y_{g(t=2)1}^{2|2}} \right]$$

where we have,

$$c_{i=1}^{1|1} = 0.056659, \quad d_{i=1}^{1|1} = 0.1, \quad h_{t=2}^{2|2} = -0.01, \quad y_{g(t=2)1}^{2|2} = -0.24$$
  
 $c_{t=2}^{2|2} = -0.223209 \quad d_{t=2}^{2|2} = 0.223209, \quad y_{g(t=2)2}^{2|2} = 0.1374$ 

and plugging in  $\nu = [1.0, 1.0, 1.0]^T$ , this yields left and right-hand side numerators of

$$g_{r_{t=2}^{2|2}}^{1|1} \left( y_{g(t=2)1}^{2|2}(\nu) + h_{t=2}^{2|2} \right) = -0.4854337 - 0.2790051j$$
  
$$g_{r_{t=2}^{2|2}}^{1|1} \left( y_{g(t=2)1}^{2|2}(\nu) - h_{t=2}^{2|2} \right) = -0.5603585 - 0.3679715j$$

and a g-value by (2.5) of

$$g_{t=2}^{2|2}(\nu) = -0.1549 + 0.1385j$$

Now we examine (3.20). Its left and right hand side numerators are

$$S\left(\hat{\lambda}_{t^{+}=2}^{2|2}(\nu)\right)^{T}\alpha_{i=1}^{2|1}, \quad S\left(\hat{\lambda}_{t^{-}=2}^{2|2}(\nu)\right)^{T}\alpha_{i=1}^{2|1}$$

where,

$$\alpha_{i=1}^{2|1} = S(\hat{\lambda}_{i=1}^{2|1}) \circ \alpha_i^{1|1}$$

plugging in  $\hat{\lambda}_i^{2|1}$ , i.e, equation (3.30e) we have

$$\begin{split} \alpha_{i=1}^{2|1} &= S([-1-1-1]^T) \circ \alpha_i^{1|1} \\ \alpha_{i=1}^{2|1} &= \begin{bmatrix} 1 & -1 & -1 & 1 & 1 & 1 & -1 \end{bmatrix}^T \circ \alpha_{i=1}^{1|1} \\ \alpha_{i=1}^{2|1} &= \begin{bmatrix} -0.66074 + 0.00000j \\ 0.000000 + 0.44709j \\ 0.000000 + 0.13438j \\ -0.00000 + -0.7705j \\ 0.023981 + 0.00000j \\ 0.137843 + 0.00000j \\ 0.013481 + 0.00000j \\ -0.00000 - 0.089900j \end{bmatrix} \end{split}$$

Now, we form  $S\left(\hat{\lambda}_{t^+=2}^{2|2}(\nu)\right)$  and  $S\left(\hat{\lambda}_{t^-=2}^{2|2}(\nu)\right)$  by using (3.30c) and get

$$S\left(\hat{\lambda}_{t^{+}=2}^{2|2}(\nu)\right) = S^{+}\left(\begin{bmatrix}1 & \operatorname{sgn}(0) & 1\end{bmatrix}\right) = S\left(\begin{bmatrix}1 & 1 & 1\end{bmatrix}\right) = \begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix}$$
$$S\left(\hat{\lambda}_{t^{-}=2}^{2|2}(\nu)\right) = S^{-}\left(\begin{bmatrix}1 & \operatorname{sgn}(0) & 1\end{bmatrix}\right) = S\left(\begin{bmatrix}1 & -1 & 1\end{bmatrix}\right) = \begin{bmatrix}1 & 1 & -1 & 1 & -1 & 1 & -1\end{bmatrix}$$

by evaluating the sgn(0) at the index  $\psi_{t=2} = (t=2)$  as a ±1, respectively. Equivalently we could have evaluated  $S_{\psi_{t=2}}^{\pm} \left( \hat{\lambda}_{t=2}^{2|2}(\nu) \right) = S \left( \hat{\lambda}_{t=2}^{2|2}(\nu) \right)$  of (3.20) by starting with  $\lambda_{t=2}^{2|2}$ of (3.31) and inserting a ±1 at index  $\psi_{t=2} = t = 2$  where the singularity has occurred (as the  $S_{\psi_t}^{\pm}(\cdot)$  symbology instructs) taking the first  $m_{i=1}^{1|1}$  elements (as the hat symbol  $\hat{\lambda}$ instructs) and then conducting the  $S(\cdot)$  expansion. A simple inner product now shows that

$$S\left(\hat{\lambda}_{t^{+}=2}^{2|2}(\nu)\right)^{T}\alpha_{i=1}^{2|1} = -0.4854337 - 0.2790051j$$
$$S\left(\hat{\lambda}_{t^{-}=2}^{2|2}(\nu)\right)^{T}\alpha_{i=1}^{2|1} = -0.5603585 - 0.3679715j$$

and a g-value of by (3.20) of

$$g_{t=2}^{2|2}(\nu) = -0.1549 + 0.1385j_{2}$$

which numerically shows the equivalence of the backward recursive and compressed characteristic functions, as well as the steps to generate the parameters of each form.

# 3.6 Term Reduction for the Compressed Characteristic Function

Equation (3.20) is in a compressed form of the parameters at step k. Therefore, any terms  $i \in [1, ..., N_t^{k|k}]$  that are similar to one another can now be added together. Terms that have equal parameterizations of their exponential argument  $y_{ei}^{k|k}(\bar{\nu})$  (for a chosen value  $\bar{\nu}$ ) can be combined by summation of the respective  $\alpha_{(\cdot)}^{k|k}$  vectors [14]. Specifically, if the parameters  $A_{(\cdot)}^{k|k}, b_{(\cdot)}^{k|k}, p_{(\cdot)}^{k|k}$  that parameterize  $y_{ei}^{k|k}(\bar{\nu})$  for any terms *i* and *j* are equal within a small numerical  $\epsilon$  value, i.e.,  $|A_{ilt}^{k|k} - A_{jlt}^{k|k}| \leq \epsilon$ ,  $|p_{il}^{k|k} - p_{jl}^{k|k}| \leq \epsilon$ , and  $|b_{ik}^{k|k} - b_{jk}^{k|k}| \leq \epsilon$ , for all  $l \in [1, ..., m_i], t \in [1, ..., n], i \neq j$ , these two (or more) terms can be combined through the addition of their vectors  $\alpha_i^{k|k}$  and  $\alpha_j^{k|k}$ . Since *j* adds with index *i* (and the term of index *j* is discarded afterwards), the  $\alpha_{(\cdot)}^{k|k}$  combine as

$$\alpha_i^{k|k} \leftarrow \alpha_i^{k|k} + \alpha_j^{k|k} \circ S\left(\tilde{\lambda}_{ij}^{k|k}\right) \tag{3.33}$$

$$\tilde{\lambda}_{ijl}^{k|k} = \operatorname{sgn}\left(\left\langle A_{il}^{k|k}, \left(A_{jl}^{k|k}\right)^T \right\rangle\right), \quad l \in [1, ..., m_i^{k|k}], \quad \tilde{\lambda}_{ij}^{k|k} \in \{\pm 1\}^{m_i^{k|k}} \tag{3.34}$$

where the hyperplanes  $A_{il}^{k|k}$  and  $A_{jl}^{k|k}$  are the rows (row-vectors) of the arrangement  $A_i^{k|k}$ and  $A_j^{k|k}$ , respectively. The symbol  $\circ$  is again used to represent the Hadamard product of elementwise vector multiplication. Note that a requirement here is that the two hyperplanes must have the same number of elements, i.e,  $m_i^{k|k} = m_j^{k|k}$ , to be tested for term reduction.

To save computational effort, the comparisons above can be restricted to the i < j condition. The characteristic function generates many similar terms at each estimation step. The process of combining terms, although it requires additional processing, was empirically shown to reduce the computational burden of the estimator tremendously, as is shown in chapter 8. It was also shown in [14, 18] that using the functional form of (3.20) leads to large computation and memory savings compared to the characteristic function of [16] because of this new ability to combine terms.

# 3.7 Dynamic Propagation Properties of the Compressed Characteristic Function

The compressed form of the characteristic function is also seen to have interesting dynamic propagation properties. If we factor out  $\alpha_i^{k|k-1}$  from (3.20), we have

$$g_{i}^{k|k}(\nu) = \frac{1}{2\pi} \left[ \frac{S_{\psi_{i}}^{+} \left(\hat{\lambda}_{i}^{k|k}(\nu)\right)^{T}}{jc_{i}^{k|k} + d_{i}^{k|k} + y_{gi}^{k|k}} - \frac{S_{\psi_{i}}^{-} \left(\hat{\lambda}_{i}^{k|k}(\nu)\right)^{T}}{jc_{i}^{k|k} - d_{i}^{k|k} + y_{gi}^{k|k}} \right] \alpha_{i}^{k|k-1}$$
(3.35)

where

$$g_{i}^{k|k}(\nu) = \left(\tilde{S}_{i}^{k|k}(\nu)\right)^{T} \alpha_{i}^{k|k-1}, \quad \bar{g}_{i}^{k|k} = \bar{\tilde{S}}_{i}^{k|k} \alpha_{i}^{k|k-1}, \\ \tilde{S}_{i}^{k|k}(\nu) \in \mathbb{C}^{s_{g}(m_{i}^{k-1|k-1}, n)}, \quad \bar{\tilde{S}}_{i}^{k|k} \in \mathbb{C}^{c_{c}(A_{i}^{k|k}) \times s_{g}(m_{i}^{k-1|k-1}, n)}.$$
(3.36)

That is,  $\tilde{S}_i^{k|k}(\nu)$  is a coefficient vector (evaluated at a chosen  $\nu$ ) of all parameters excluding  $\alpha_i^{k|k-1}$  in (3.35), and  $\bar{S}_i^{k|k}$  is a coefficient matrix where the vector  $\tilde{S}_i^{k|k}(\nu)$  is evaluated in each cell  $\nu \in C_j$ ,  $j \in [1, ..., c_c(A_i^{k|k})]$  and stacked row-wise. However, we know that the g-function can also be written as

$$g_i^{k|k}(\nu) = S\left(\lambda_i^{k|k}(\nu)\right)^T \alpha_i^{k|k} = \left(\tilde{S}_i^{k|k}(\nu)\right)^T \alpha_i^{k|k-1},\tag{3.37}$$

and written for all cells as

$$\bar{g}_{i}^{k|k} = \bar{S}_{i}^{k|k} \alpha_{i}^{k|k} = \bar{\tilde{S}}_{i}^{k|k} \alpha_{i}^{k|k-1}, \qquad (3.38)$$

where

$$\alpha_i^{k|k-1} = S\left(\hat{\lambda}_i^{k|k-1}(\nu = H_k^T)\right) \circ \alpha_i^{k-1|k-1} = \operatorname{diag}\left(S\left(\hat{\lambda}_i^{k|k-1}(\nu = H_k^T)\right)\right) \alpha_i^{k-1|k-1}.$$

It is clear then that  $\alpha_i^{k|k}$  and  $\alpha_i^{k-1|k-1}$  are linearly related as

$$\alpha_i^{k|k} = \left(\bar{S}_i^{k|k}\right)^{\dagger} \left(\bar{\tilde{S}}_i^{k|k}\right) \operatorname{diag}\left(S\left(\hat{\lambda}_i^{k|k-1}(\nu = H_k^T)\right)\right) \alpha_i^{k-1|k-1}$$
(3.39)

which forms a dynamic propagation formula from step k-1 to step k for the parameter  $\alpha$ . A dynamic propagation property for the  $\alpha$  parameters (3.6), (3.23) and (3.39) was seen in [13] for the scalar Cauchy estimator, but only now has a closed-form dynamic propagation equation been discovered for these parameters in the multivariate case. Remember that the propagation equations of (3.23) and (3.17) rely upon computing the sign vector of (3.22c), which sets  $\nu = H_k^T$ .

Furthermore, an inspection of the propagation of the parameter set  $\{A_i^{k|k}, p_i^{k|k}, q_i^{k|k}, d_i^{k|k}, \lambda_i^{k|k}, \lambda_i^{k|k}$  indicates that if the system dynamics of (2.1) are LTI, these parameters can be computed completely offline. This is because they are *not* functions of the measurement  $z_k$ . If the initial conditions  $\{A^{1|0}, p^{1|0}, b^{1|0}\}$  are changed, however, the generated parameter sets will not be the same unfortunately. Therefore, these parameters can only be calculated 'offline' for repeated use if and only if the provided initial conditions are assumed never to change<sup>3</sup>. The parameters  $\{\alpha_i^{k|k}, c_i^{k|k}, b_i^{k|k}\}$  are, however, functions of the measurement  $z_k$ . If the initial conditions do not change, the only (online) computational cost of the estimator would be to solve for the parameter set  $\{b_i^{k|k}, c_i^{k|k}\}$ , evaluate  $\bar{g}_i^{k|k}$ , and solve the system of equations (3.5) for each term's  $\alpha_i^{k|k}$ . The only remaining work is to sum the  $\alpha_{(\cdot)}$  vectors for terms that are found to reduce with one another within term reduction (the next paragraph elaborates more on this). Under these assumptions, the run-time of the estimator would be extremely fast.

An interesting insight was made in [22], regardless of the initial conditions provided to the MCE or the measurement realization. Based on the theory presented in [22] for LTI systems, it is *a-priori* known which hyperplanes of a term will coalign with one another. Moreover, it is *a-priori* known which terms at each step k will reduce with one another. This can save a large amount of computation, particularly as the term reduction algorithm has a quadratic run-time. If the system is LTI, it is advised to run the MCE algorithm once offline and store the indices where hyperplanes of a term coalign as well as the indices where terms reduce together at each estimation step. For all future runs, this information can be loaded into memory and re-used to reduce the computational effort in

<sup>&</sup>lt;sup>3</sup>This is seen not to be the case in chapter 5, where the proposed strategy to initialize the MCE will change its initial set of conditions each time the algorithm is reset.

the coalignment sub-routine. In the term reduction sub-routine, the quadratic run-time search to find which terms reduce together can be removed completely. Terms at these a-priori known indices can now simply be added together. Effectively, this turns the term reduction algorithm from quadratic to a linear one in the number of term that reduce. This is demonstrated in section 8.1. It should be noted that for this reason, the savings will be much larger in the term reduction sub-routine then it is in term-coalignment.

Although the term reduction algorithm tests the  $b_i^{k|k}$  terms for equality, which is indeed a function of the measurement  $z_k$ , it is luckily a linear function of the measurement. Therefore, we could simply set  $z_k$  to any fictitious value, i.e  $z_{(\cdot)} = 1$  in the offline stage, store the indices at which terms reduce together, and use these indices nevertheless in the online stage. These facts hold true regardless of the measurement or the initial conditions provided to the MCE.

#### 3.7.1 Discarding Negligible Terms

A numerical study of the terms of the characteristic function reveals that many of the  $\alpha_i^{k|k}$  parameters approach zero as the estimation step k increases. This is significant, as the  $\alpha$  parameters are used to evaluate (3.20). As the conditional mean and covariance are a function of the values of (3.20) or (2.5), this observation indicates if  $||\alpha_i^{k|k}|| \to 0$ , the value of  $g_i^{k|k}(\nu) \to 0$  for (3.20), and the contribution of the term towards the conditional mean and covariance will be negligible. Furthermore, it is observed from (3.39) that if a parent term has its  $||\alpha_i^{k|k}|| \to 0$ , its generated child terms will have a norm close to zero as well. This implies that removing any  $\alpha_i^{k|k}$  which has this property would remove a factorial number of terms at future estimation steps k.

Formally, the dynamic propagation matrix of (3.39), written shorthand as  $S_i$  for  $\alpha_i^{k|k} = S_i \alpha_i^{k-1|k-1}$  could be decomposed by the singular value decomposition  $S_i = U \Sigma V^T$ and the singular values  $\Sigma$  can be examined. If all singular values of the matrix  $S_i$  are less than 1, then we can be certain that the propagated alpha-vector will have a norm smaller than that of its parent. This type of matrix is considered a contractor, as any vector multiplied by this matrix will reduce its norm. A numerical study reveals that, in general, the singular values of this matrix are usually less than 1, however, it cannot be guaranteed in general. Moreover, calculating the singular values of the matrix in the online setting would be an expensive procedure, as the dimensions of the matrix  $S_i$  can be large.

Instead of computing this matrix directly, we could wait for the  $||\alpha_i^{k|k}||$  to fall well below or close to the value of machine precision, which for double floating point numbers is 1e-16. In numerical testing, the value of  $\epsilon$  was set between 1e-20 and 1e-14. Since  $\alpha$  is a vector of complex numbers, testing whether the 1-norm over the vector is less than  $\epsilon$  is an appropriate metric. It was observed that removing the terms of the estimator whose  $\alpha_i^{k|k}$  fell within this range had negligible impact on the state estimates. The number of terms, however, reduced significantly. This is shown in further detail in the experiments of section 8.1. The main finding was that: 1.) computing the  $\alpha$ -vectors have allowed similar terms to reduce together and 2.) provided a means to identify terms of the characteristic function that have a negligible contribution. Removing negligible terms before they generate new (negligible) child terms at the next estimation step is seen to increase the speed of the estimator, and in some cases, significantly.

## Chapter 4

# An Efficient Algorithm for Compressing the G-Function

In this chapter, we show that using several insights presented in [23], it is possible to compute the parameters  $\alpha_i^{k|k}$  without the need to solve an expensive cell-enumeration problem and furthermore without ever having to take a pseudo-inverse of a large matrix, as done in chapter 3. The algorithm presented in this chapter was first presented by the author of this dissertation in [18].

### 4.1 Preliminaries

We first address several differences between components of the characteristic function in chapter 3 and the structure of an algorithm based on the results in [23] would require to construct the  $\alpha$ -vectors. First, the theory in [23] makes use of indicator functions  $\sigma$ and not sign functions sgn as (3.20) does. This can be fixed by relating one to the other as  $\sigma(\cdot) = \frac{1}{2}(\operatorname{sgn}(\cdot) + 1)$ . Next, the theory in [23] deals with hyperplane arrangements in a general position  $A\nu = b, A \in \mathbb{R}^{m \times n}$  where the offset from the origin  $b \neq \mathbf{0}^m$ . The hyperplane arrangements of the characteristic function for the MCE, however, are in central position  $A\nu = \mathbf{0}$ . This is easily circumvented by slightly perturbing at least m - n right-hand side elements of the zero vector to non-zero values. This is allowable since shifting hyperplanes of a central arrangement away from the origin will preserve the
original (exterior) sign-vectors of the cells, only adding new (interior) cells. Moreover, the theory in [23] also makes the assumption that no hyperplane in the arrangement has a normal aligned with  $e_n = [0, ..., 1]^T \in \mathbb{R}^n$ . If an arrangement happens to have such a hyperplane, a simple rotation of the arrangement will resolve this issue. Lastly, it is possible for general and central arrangements to have less than either  $s_g(m, n)$  or  $s_c(m, n)$  cells as given by (3.1) and (3.2), respectively. Specifically, a degenerate central arrangement is one where the number of cells counted by an enumeration algorithm is less than  $s_c(m_i^{k|k}, n)$  and a degenerate general arrangement is one where the number of cells counted is less than  $s_g(m_i^{k|k}, n)$  [20]. This is due to the various geometrical patterns that can arise in both central and general arrangements (see Fig. 3.1). This problem is resolved in section 4.2.5.

### 4.2 Procedure for the Efficient Computation of $\alpha_i^{k|k}$

For brevity, in what follows, we drop the (term) subscript i and (step) superscript k|k on the hyperplane arrangement  $A_i^{k|k}$  and  $g_i^{k|k}(\nu)$ , and use i as a general indexing variable. For a general arrangement H of m hyperplanes in n dimensions, let  $H_i = \{\nu \in \mathbb{R}^n | a_i^T \nu = \tilde{b}_i\}, i \in [1, ..., m]$  represent the hyperplanes of the arrangement. Here,  $a_i \in \mathbb{R}^n$  is the normal to  $H_i$  that is stored row-wise in  $A \in \mathbb{R}^{m \times n}$  and  $\tilde{b}_i \in \mathbb{R}$  is an element of  $\tilde{b} \in \mathbb{R}^m$ . In the MCE context, it is a perturbation applied to shift the hyperplanes  $\{\nu \in R^n | a_i^T \nu = 0\}$ into a general position. The hyperplanes define open halfspaces  $H_i^+ = \{\nu \in \mathbb{R}^n | a_i^T \nu > \tilde{b}_i\}$ and  $H_i^- = \{\nu \in \mathbb{R}^n | a_i^T \nu < \tilde{b}_i\}$ . Let  $\sigma_i(\nu) = \frac{1}{2}(\operatorname{sgn}(a_i^T \nu - \tilde{b}_i) + 1)$  be an indicator function over the open halfspaces of  $H_i$ , which is equal to 1 for all  $\nu \in H_i^+$  and 0 otherwise. Note that cells of A are defined by at least n halfspaces of  $H_i$ . The compressed form of the characteristic function in the MCE presented in chapter 3 is based on the following result.

**Theorem 1** (From [23]). Let A be a hyperplane arrangement of m affine hyperplanes  $H_i, i \in 1, ..., m$  in  $\mathbb{R}^n$  and  $\sigma_i(\nu)$  be the indicator function of the open halfspaces of  $H_i$ . Since  $g(\nu)$  is a function that is constant in the interior of every cell, there is a linear combination of products of n or less of the functions  $\sigma_i(\nu)$  that is equal to  $g(\nu)$  at any point not in  $\cup_{i=1}^{m} H_i$ .

Theorem 1 states that, much like (3.36), we can construct the values of  $g(\nu)$  by a linear combination using products up to n out of the m indicator functions  $\sigma_i(\nu)$ . Specifically, let  $I \subset I_m$ , where  $I_m = \{1, ..., m\}$ . Let  $\sigma_I(\nu) = \prod_{i \in I}^n \sigma_i(\nu)$ , which is the product of the indicators within set I. We define  $\sigma_{\emptyset} = 1$  for the empty set |I| = 0. Thus, theorem 1 implies

$$g(\nu) = \sum_{|I| \le n} \alpha_I \sigma_I(\nu), \qquad (4.1a)$$

or, equivalently,

$$g(\nu) = \sum_{|I|=n} \alpha_I \sigma_I(\nu) + \sum_{|I|< n} \alpha_I \sigma_I(\nu).$$
(4.1b)

Note that the maximum number of possible combinations of  $|I| \leq n$  is given by  $s_g(m,n)$  of (3.2). Also note that writing  $\sigma_i$  or  $\sigma_I$  is shorthand for  $\sigma_i(\nu)$  or  $\sigma_I(\nu)$ , respectively. Our goal is to efficiently find the coefficients  $\alpha_I$  of (4.1) that can recreate the values of  $g(\nu)$  for any  $\nu \in \mathbb{R}^n$ . We wish to do so by exploiting the unique properties of functions, such as (3.20), that are constant within the cells of its hyperplane arrangements. To develop an algorithm to do this, we first look at (4.1b) and focus on the coefficients for |I| = n.

### 4.2.1 Finding the Coefficients of |I| = n

We start by finding the vertices of the arrangement H. The intersection point  $x_I$  of hyperplanes indexed by  $I \subset I_m$  with cardinality |I| = n are those vertices. An arrangement in general position has  $\binom{m}{n}$  such subsets and hence vertices. The n hyperplanes that intersect at  $x_I$  partition  $\mathbb{R}^n$  into  $2^n$  regions. Since no hyperplane has its normal aligned with  $e_n$ , i.e.,  $\langle a_i, e_n \rangle \neq 0$ ,  $\forall i \in I_m$ , for a particular region,  $x_I$  is the lowest point in the region w.r.t  $e_n$ . Moreover, the region where this occurs corresponds to a particular cell in the arrangement. Let  $\{C_1, ..., C_{2^n}\}$  be the set of  $2^n$  cells that encircle  $x_I$  and let  $C_{x_I} \in \{C_1, ..., C_{2^n}\}$  be the cell for which  $x_I$  is the lowest point.  $C_{x_I}$  is referred to as the upper cell of  $x_I$  and finding a point in it is the topic of section 4.2.2.

The efficient computation method of  $\alpha_I$  in [23] requires that  $\sigma_i(\nu) = 1$  for all  $\nu \in C_{x_I}$ 

and  $i \in I$ . If this is not the case, we define  $I' \subset I$  to be a set of all i such that if  $\sigma_i(\nu \in C_{x_I}) = 0$ , then  $\sigma'_i(\nu) = (1 - \sigma_i(\nu))$ , and

$$\sigma_I'(\nu) = \prod_{i \in I'} \sigma_i'(\nu) \prod_{i \in I \setminus I'} \sigma_i(\nu).$$
(4.2)

Note that  $\sigma'_{I}(\nu) = 1$  for  $\nu \in C_{x_{I}}$  and  $0 \forall \nu \in \{C_{1}, ..., C_{2^{n}}\} \setminus C_{x_{I}}$ . Using (4.2), (4.1b) is restated as

$$g(\nu) = \sum_{|I|=n} \alpha'_I \sigma'_I(\nu) + \sum_{|I|< n} \alpha'_I \sigma'_I(\nu).$$

$$(4.3)$$

To compute the  $\alpha'_I$  for |I| = n, we define

$$S_{I}(\nu) = \prod_{i \in I'} (2\sigma'_{i}(\nu) - 1) \cdot \prod_{i \in I \setminus I'} (2\sigma_{i}(\nu) - 1).$$
(4.4)

It is easy to verify that  $S_I(\nu) \in \{-1, 1\}$  for all  $\nu$  and equals 1 for  $\nu \in C_{x_I}$ . Moreover, it has opposing signs when evaluated at  $\nu$  in neighboring cells, i.e., two cells that share a hyperplane. Then, for all |I| = n [23]

$$\alpha_I' = \sum_{i=1}^{2^n} g(\nu) S_I(\nu) \Big|_{\nu \in C_i}.$$
(4.5)

Equation (4.5) states that to find the coefficient  $\alpha'_I$  for a set of hyperplanes I, we need to evaluate  $g(\nu)$  on the  $2^n$  cells that encircle its vertex  $x_I$ . We can find the value of gin these cells by determining the sign-vectors  $\lambda$  with respect to the entire arrangement H (see (3.20)) as follows. First, for  $\{H_i, i \in I_m \setminus I\}$ , i.e., hyperplanes that do not include the vertex  $x_I$ , we compute  $\operatorname{sgn}(a_i^T x_I - \tilde{b})$ . Second, the sign-vector for the upper cell  $C_{x_I}$ is formed by conjoining values of 1 for  $i \in I$  to the sign sequences computed above. The sign vectors of the remaining  $2^n - 1$  cells are obtained by permuting the signs of the indexes  $i \in I$ .

The time-complexity of the operations presented is  $\mathcal{O}(n^3)$  to find each vertex point  $x_I$  and  $\mathcal{O}(2^nmn)$  to construct the sign-vectors for the  $2^n$  cells encircling  $x_I$ . The only expense left is the  $2^n$  evaluations of  $g(\nu)$  in (3.20), *i.e*  $\mathcal{O}(2^ng(\nu))$ . The aforementioned

steps are repeated over the  $\binom{m}{n}$  total vertices. The coefficients  $\alpha'_I$  for |I| = n are now constructed. See section 4.2.4 for the procedure to transform the coefficients  $\alpha'_I$  back to  $\alpha_I$ .

### 4.2.2 Finding a Point in the Upper Cell of a Vertex

Here we present an algorithm that can find a point in the upper cell of a vertex for any dimension  $d \leq n$ . Section 4.2.3 clarifies why this algorithm needs to work for any  $d \leq n$ , but for now we can think of d = n. We define the upper cell for a vertex  $x_I$ (and |I| = d) as the cell for which vertex  $x_I$  is the lowest point, with respect to the coordinate direction  $e_d$ . The algorithm presented below is used within the procedures of sections 4.2.1 and 4.2.3.

Given a set I of d-dimensional hyperplanes, with  $|I| = d \leq n$ , a generalized cross product can be used to construct a point in the upper cell of the hyperplanes in I. This will find the directions that are parallel to each edge formed by the intersection of d-1 hyperplanes in I. That is, for all  $I_{d-1} \subset I$  and  $|I_{d-1}| = d$ -1, we compute the generalized cross product direction  $p_i, i \in \{1, ..., d\}$ . These directions are checked to make sure they point upwards (w.r.t  $e_d$ ), i.e., that their last component is positive, and are flipped if not. A point in the upper cell of  $x_I$  is thus  $p_{x_I} = x_I + \sum_{i=1}^d p_i$ .

The general time-complexity is  $\mathcal{O}(d^5)$ . This is because a determinant of matrix size  $(d-1 \times d-1)$  takes time-complexity of order  $\mathcal{O}(d^3)$ , which is repeated d times for a single edge, and for all d directions (edges) of a vertex I.

### 4.2.3 Finding the Coefficients of |I| < n

In this subsection, we first show how the coefficients for d = |I| = n-1 are found, followed by comments to generalize the procedure to d = |I| < n-1.

To compute the coefficients of |I| = n-1, we start by finding the vertex  $x_I$  computed for all |I| = n that has a minimum value in the direction  $e_n$ . Let us refer to this vertex as  $x_I^m$ . Next we construct a hyperplane  $H_L = \{\nu \in \mathbb{R}^n | \langle e_n, \nu \rangle = b_L\}$ , with  $b_L = e_n^T x_I^m - \epsilon$ and  $\epsilon \in \mathbb{R} > 0$ , i.e.,  $H_L$  is a horizontal hyperplane that is below  $x_I^m$ . Therefore,  $H_L$  is also below all upper cells  $C_{x_I}$  for all  $I \subset I_m$  and |I| = n, and hence  $\sigma'_I(\nu) = 0$  for all  $\nu \in H_L$ . Consequently, when evaluated for  $\nu \in H_L$ ,  $g(\nu) = \sum_{|I| < n} \sigma'_I(\nu) \alpha'_I$ , or alternatively

$$g(\nu) = \sum_{|I|=n-1} \alpha'_{I} \sigma'_{I}(\nu) + \sum_{|I|< n-1} \alpha'_{I} \sigma'_{I}(\nu).$$
(4.6)

Projecting the hyperplanes  $H_1, ..., H_m$  onto  $H_L$  will result in new hyperplanes  $\bar{H}_i = \{\bar{\nu} \in \mathbb{R}^{n-1} | \bar{a}_i^T \bar{\nu} = \bar{b}_i = \tilde{b}_i - a_i[n] b_L\}, i \in [1, ..., m]$  where  $\bar{a}_i \in \mathbb{R}^{n-1} = a_i[1 : n-1]$  and  $a_i[n] = e_n^T a_i$ . This projection yields a  $\mathbb{R}^{n-1}$  dimensional subspace of  $\nu \in \mathbb{R}^n$  and the coefficients  $\alpha'_I$  for |I| = n - 1 can be computed using the procedure detailed in the previous subsection. This process can be similarly applied for lower dimensions d < n-1.

The time-complexities for the lower dimensions  $d \leq n$  would follow very closely to that of sections 4.2.1 and 4.2.2 by replacing n with d. The additional work here includes the  $\binom{m}{d}$  comparisons required to find the minimum vertex, which does not require any floating point operations. Moreover, projecting the arrangement down a dimension requires only  $\mathcal{O}(m)$  operations to update the affine offsets. Note that, to leading order for  $\mathcal{O}(\cdot)$ , the operations at d = n will dominate those of  $d \leq n-1$ .

### 4.2.4 Unpriming the Coefficients of $|I| \le n$

After computing all  $\alpha'_I$  using the basis of  $\sigma'_I$ , we are ready to discuss the procedure to recover the coefficients  $\alpha_I$  for  $|I| \leq n$  in the original basis  $\sigma_I$  used in (4.1). For all  $\sigma_i(p_{x_I}) \neq 1$  and  $i \in I$ , the parenthesis of  $(1 - \sigma_i)$  that are the result of any  $i \in I' \subset I$ in (4.2) must now be opened to transform the coefficients  $\alpha'_I$  of (4.3) back to the  $\alpha_I$  of (4.1). This is necessary to keep the indicator directions  $\sigma_i$  consistent across all projection steps of section 4.2.3. This is also necessary to keep the indicator directions consistent with the orientation of the basis vectors in (3.20).

We call this process of opening the parenthesis and transforming the coefficients  $\alpha'_I$ of (4.3) to  $\alpha_I$  of (4.1a) *unpriming*. A simple example can help illustrate the unpriming procedure to transform  $\alpha'_I$  back to  $\alpha_I$ . For an arrangement of m = 2 hyperplanes of dimension n = 2, using (4.3), the coefficient g is expressed as

$$g = \alpha'_{\emptyset}\sigma_{\emptyset} + \alpha'_{1}\sigma'_{1} + \alpha'_{2}\sigma'_{2} + \alpha'_{12}\sigma'_{12}, \qquad (4.7)$$

where  $\alpha'_{12}$  and  $\sigma'_{12}$  are shorthand for  $\alpha'_{I=\{1,2\}}$  and  $\sigma'_{I=\{1,2\}}$ , respectively. Suppose that when constructing  $\alpha'_{12}$  for  $I = \{1,2\}$ ,  $\sigma_{12}(p_{x_{12}}) = 0$  due to  $\sigma_1$ . Hence, to use (4.5), we formed  $\sigma'_{12} = (1 - \sigma_1)\sigma_2$  and thus  $I' = \{1\} \subset I$  for (4.2). Similarly, when computing  $\alpha'_2$ for  $I = \{2\}$  it was found that  $\sigma_2(p_{x_2}) = 0$  and thus was replace by  $\sigma'_2 = (1 - \sigma_2)$  with the related  $I' = \{2\} \subset I$ . Moreover,  $\sigma_1$  was not changed computing  $\alpha'_1$ . Consequently, (4.7) is manipulated as follows:

$$g = \alpha'_{\emptyset}\sigma_{\emptyset} + \alpha'_{1}\sigma_{1} + \alpha'_{2}(1 - \sigma_{2}) + \alpha'_{12}(1 - \sigma_{1})\sigma_{2}$$
  
=  $(\alpha'_{\emptyset} + \alpha'_{2})\sigma_{\emptyset} + \alpha'_{1}\sigma_{1} + (\alpha'_{12} - \alpha'_{2})\sigma_{2} + (-\alpha'_{12})\sigma_{12}.$  (4.8)

It is clear that, in the original basis of  $\sigma_I$ ,  $\alpha_{\emptyset} = (\alpha'_{\emptyset} + \alpha'_2)$ ,  $\alpha_1 = \alpha'_1$ ,  $\alpha_2 = (\alpha'_{12} - \alpha'_2)$  and  $\alpha_{12} = -\alpha'_{12}$ . In general, the above reveals that unpriming the coefficients of  $\sigma'_I$ -s with non-empty I' will add or subtract  $\alpha'_I$  from coefficients of  $\sigma_J$  with |J| < |I|.

The worst-case time complexity is when all  $\max\{\binom{m}{d}, d = [1, ..n]\}$  coefficients  $\alpha'_I$  at dimension |I| = d have an |I'| = d and therefore need to add their  $\alpha'_I$  to the  $\sum_{i=0}^{d-1} \binom{m}{i}$  coefficients  $\alpha_I \in \alpha$  where  $|I| \leq d$ -1. This results in (worst case)  $\mathcal{O}(\binom{m}{d})$  additions needed for a particular  $\alpha'_I$  and where d is chosen using the criteria above. For the whole vector  $\alpha$ , the unpriming step would add  $\mathcal{O}(\binom{m}{d}^2)$  additional operations.

### 4.2.5 Finding the Coefficients of Arrangements with Degeneracies

As mentioned in section 4.1, due to various geometrical patterns, both general and central hyperplane arrangements can have fewer cells than (3.1) and (3.2) suggest. The procedures given in sections 4.2.1 to 4.2.4 assume that these degeneracies are not present and that the arrangement  $A, \tilde{b}$  contains the number of cells given by (3.2). However, degeneracies are seen to occur in the arrangements generated by the MCE.

An arrangement is degenerate if there are (one or more) sets  $\{I \in I_m \mid |I| = n\}$  of hyperplanes with normal vectors that are linearly dependant, i.e.,  $\operatorname{rank}(A_I) < n$ . Consequently, there is no vertex  $x_I$  defined by those n hyperplanes, as  $x_I = A_I^{-1}\tilde{b}_I$  does not exist. This, however, is not an issue. For any set of hyperplanes I seen to be rank deficient (or a condition number deemed too large), the corresponding coefficient  $\alpha_I$  can simply be set to zero. This implies that since there exists no vertex, the coefficient corresponding to this vertex is not required. This holds true for any dimension, see [23].

There is no added expense in the routine due to degeneracy, except drawing m random (small) floating point numbers to declare a  $\tilde{b}$ . Solving for each vertex is an  $\mathcal{O}(n^3)$ procedure. Therefore, the complexity for evaluating a single combination I, is still  $\mathcal{O}(n^3)$ .

### 4.2.6 Time-Complexity of the Proposed Algorithm

We have now described all parts of the proposed algorithm to compute the vectors  $\alpha$  for (3.20). The total time-complexity of the procedure for a single coefficient  $\alpha_I$  is the sum of the complexities presented in sections 4.2.1 to 4.2.5. Computing a coefficient with a dimension d = n sums to  $\mathcal{O}(2^n g(v) + n^3 + 2^n mn) + \mathcal{O}(n^5) + \mathcal{O}(m) + \mathcal{O}(\binom{m}{n})$ . To obtain the complexity of evaluating the function  $g(\nu)$  of (3.20), note that  $g(\nu)$  has two dot products between vectors of length  $s_g(m - r, n)$  in the numerator, with  $s_g(m - r, n)$  given by (3.2), as well as three additions, two subtractions, and two divisions. To leading order, an evaluation of  $g(\nu)$  will take  $\mathcal{O}(\binom{m}{n})$  operations. Therefore, to the leading order,  $\mathcal{O}(2^n g(\nu)) = \mathcal{O}(2^n \binom{m}{n})$ . Each  $\alpha$ -vector that is constructed contains  $s_g(m, n) = \sum_{i=0}^d \binom{m}{n}$  coefficients and to leading order,  $\binom{m}{n}$ . Furthermore,  $\binom{m}{n}$  can be bounded above by  $m^n$ . Putting this all together we see the time-complexity to construct the vector  $\alpha$  will consist of repeating  $\mathcal{O}(2^n m^n)$  operations  $\mathcal{O}(m^n)$  times. Thus, our final result is that the procedure takes on the order of  $\mathcal{O}(2^n m^{2n})$  operations to construct the  $\alpha$ -vector.

Let us compare this result to the procedure proposed in [14] to compute  $\alpha$ . As noted in chapter 3, the procedure of [14] involves solving both a cell enumeration problem as well as the large linear system in (3.36). For a (non-degenerate) cell enumeration problem of m central hyperplanes of dimension n,  $2^{m-1}$ -1 linear programs are solved. The timecomplexity of a linear program depends on the underlying method used to reach the optimum. For example, the simplex method developed by Dantzig [27] was shown to have worst-case complexity of  $\mathcal{O}(n^{2}2^{n})$  in [28]. For interior-point algorithms, methods have been proposed with the complexity of  $\mathcal{O}(n^{3.5}L)$  [29] and  $\mathcal{O}(((m+n)n^{2}+(m+n)n^{1.5})L)$ [30] where  $\mathcal{O}(L)$  is the number of bits of precision required. These bounds are presented for completeness but are quite theoretical. Still, they are used to show concretely the advantage of the method proposed in the current study. The pseudo-inverse in (3.6) requires  $\mathcal{O}(s_{c}(m,n)^{2}s_{g}(m,n)) = \mathcal{O}((2\sum_{i=0}^{n-1} {m-1 \choose i})^{2}(\sum_{i=0}^{n} {m \choose i}))$  operations. Substituting  $m^{n}$  to bound  ${m \choose n}$ , the method in [14] has the computational complexity of  $\mathcal{O}(m^{3n})$ . Compared to the newly proposed algorithm we conclude that  $\mathcal{O}(2^{n}m^{2n}) \leq \mathcal{O}(m^{3n})$  for  $m \geq 2$ . Regardless of the LP technique chosen in the scheme of [14], the proposed procedure here is superior (and can even be several magnitudes faster), as either m or nare increased.

### 4.2.7 Converting Between the Indicator and Sign Basis

The vector  $\alpha$  is constructed for indicator basis functions. To use  $\alpha$  in (3.20), one must convert  $\alpha$  to the sign basis. Let us temporarily denote the indicator basis vector as  $\alpha^{I} \in \mathbb{C}^{s_{g}(m,n)}$  and the indicator basis matrix as  $S^{I} \in \{0,1\}^{s_{g}(m,n) \times s_{g}(m,n)}$ . These can be related to the sign basis vector  $\alpha^{S} \in \mathbb{C}^{s_{g}(m,n)}$  and matrix  $S^{S} \in \{\pm 1\}^{s_{g}(m,n) \times s_{g}(m,n)}$ through (3.36) as  $\bar{g} = S^{I} \alpha^{I} = S^{S} \alpha^{S}$ , where  $\alpha^{S}$  is the desired vector within the sign-basis. Therefore,

$$\alpha^S = X_I^S \alpha^I, \tag{4.9}$$

where  $X_{I}^{S} = (S^{S})^{-1}S^{I}$ .

A simple example illustrates how to construct  $S^I$  and  $S^S$ . For m = n = 3, place all combinations of up to n and out of the m elements of the general vector  $\lambda = [\sigma_1, \sigma_2, \sigma_3]$ row-wise in the matrix  $\Lambda \in \mathbb{R}^{s_g(m,n) \times n}$ . Now,  $S^I$  and  $S^S$  are formed by the row-wise expansion of  $\Lambda$  using the  $S(\cdot)$  basis expansion function and by setting all  $\sigma_{(\cdot)} = 1$  below

$$\Lambda = \begin{bmatrix} 0 & 0 & 0 \\ \sigma_{1} & 0 & 0 \\ 0 & \sigma_{2} & 0 \\ 0 & 0 & \sigma_{3} \\ \sigma_{1} & \sigma_{2} & 0 \\ \sigma_{1} & 0 & \sigma_{3} \\ 0 & \sigma_{2} & \sigma_{3} \\ \sigma_{1} & \sigma_{2} & \sigma_{3} \end{bmatrix}, \quad S^{I} = S(\Lambda), \quad S^{S} = S(2\Lambda - 1).$$
(4.10)

Although the basis matrices  $S^I, S^S$  are integer-valued,  $X_I^S = (S^S)^{-1}S^I$  is not. The added time-complexity associated with (4.9), which is on the order of  $\mathcal{O}(m^{2n})$ , is negligible compared to the other components of the proposed algorithm. Moreover, note that the matrix  $X_I^S$  is not a function of the measurements. Therefore it can be precomputed for any desired *m* offline and stored before the online execution of the MCE to further enhance its real-time implementation.

### 4.2.8 Comments on the Proposed Implementation

To conclude this section, we remark that a large amount of computation when forming the coefficients  $\alpha_I$  (using (4.5)) can be saved by utilizing an efficient caching data structure. Note that coefficients  $\alpha_I$  that have been formed by (4.5) will share common  $g(\nu)$  values with other coefficients  $\alpha_J, J \neq I$ . Specifically, for a vector  $\alpha$  of length  $s_g(m, n)$ , a total of  $N_{eval} = \binom{m}{n} 2^n + \binom{m}{n-1} 2^{n-1} + \ldots + \binom{m}{1} 2^1$  evaluations of  $g(\nu)$  are required. However, there are only  $s_g(m, n) \ll N_{eval}$  values of  $g(\nu)$  - one per cell - after perturbing the central arrangement into a general position. If a caching data structure is used to store the  $s_g(m, n)$  values of  $g(\nu), N_{eval} - s_g(m, n)$  evaluations of  $g(\nu)$  can be avoided by instead looking up the value in the cache. Since each cell, C has a unique sign-vector, and since  $g(\nu)$  is constant in a cell, a specific value of  $g(\nu)$  value can be retrieved from the cache by its respective sign-vector. Therefore, an associative data structure between the sign-vector and  $g(\nu)$  value can be employed as the caching mechanism, such as a dictionary.

When computing (4.5), the run-time performance will benefit greatly if the time required to hash the sign-vector and retrieve its cached g-value is less than the time required to re-compute the g-value from scratch.

## Chapter 5

### The Sliding Window Approximation

# 5.1 Derivation of the Sliding Window Approximation

Although the compressed structure of the characteristic function reduces memory consumption and allows for similar terms to be combined, the number of terms after a measurement update will still grow, albeit now more slowly. A method termed the sliding window approximation, was proposed in [31] for two-state linear systems to cap the growth rate and run the estimation structure continuously with a fixed computation load per estimation step. The method is explained in this section and generalized to multivariate systems.

To run the MCE continuously and for arbitrarily long simulations, a bank of W estimators processing data over sliding windows (*i.e.*, MCEs) is instantiated. Each estimator has a processing capacity of W sensor measurements. Clearly, W dictates the computational load of the filter bank and is set a-priori to account for the computing power at hand. The data windows are staggered by one estimation step. Only the estimator that processed W sensor measurements at a specific time step k will report its estimation result, following which it will be restarted using the procedure detailed below. Subsequently, the neighboring estimator that has processed W measurements at time step k+1will report its mean and covariance at that instant. Figure 5.1 illustrates the proposed schematic for an example of six estimators and thus six sliding windows.

In [31], a formula was derived to initialize the characteristic function of two-state linear systems to generate a *desired* mean and covariance, given the measurement value, over one estimation step. This formula was useful for the MCE for a two-state system using the sliding window approximation. It allowed the "restarted" window to reconstruct the estimate found by the "full" MCE window of W measurements at each estimation step. In this way, each restarted window would be initialized about the current estimate that is computed using the previous W measurements. The initialization formula, however, was not generalizable to the multivariate case. This issue is resolved next.



#### Sliding Window Bank Example of Six Estimators

Figure 5.1: Schematic of the sliding window bank for six estimators (windows) each a function of six sensor measurements.

### 5.1.1 Sliding Window Initialization for Multivariate Systems

The procedure for initializing the MCE using the sliding window approximation is now developed explicitly. Let the window length be W and therefore there are also W windows. After processing W measurements in window  $w \in W$  at time step k, the associated MCE computes the resulting conditional mean and error variance. Now, this MCE has to be initialized before it processes the next measurement at k + 1. It is suggested that this initialization is constructed such that after processing the measurement at k + 1, the resulting conditional mean and error variance are the same as those computed by the neighboring MCE that has processed W measurements at this time instance. We start with the form of the characteristic function, before the initial measurement update, given as

$$\bar{\phi}_{X_1} = \exp\left[\left(-\sum_{\ell=1}^n p_\ell^1 \left|\langle a_\ell^1, \nu \rangle\right|\right) + j\langle b^1, \nu \rangle\right]$$
(5.1)

with initialization parameters  $a_{\ell}^{1} \in \mathbb{R}^{n}, p_{\ell}^{1} \in \mathbb{R}, b^{1} \in \mathbb{R}^{n}$ . The initial set of hyperplanes  $A = [a_{1}^{1}, a_{2}^{1}, ..., a_{n}^{1}]^{T}$  should be chosen as orthonormal vectors, where  $A^{-1} = A^{T}, AA^{T} = I$ ,  $\det(A) = 1$ , but not necessarily as unit vectors as in [16]. Given the constraints on A, the key idea is that the parameter set  $\{A, p^{1}, b^{1}\}$  is now chosen such that updating the characteristic function of (5.1) with a single measurement  $z \in \mathbb{R}$  (via a measurement update) will generate a conditional mean  $\hat{x}$  and covariance P equal to that of the neighboring MCE of W measurement updates. The transformation A is used on the spectral vector as  $\nu = A\bar{\nu} \Rightarrow \bar{\nu} = A^{T}\nu$ . Then, the characteristic function of (5.1) reduces to a form similar to that used in section 5.3 in [16] as

$$\phi_{X_1} = \exp\left[\left(-\sum_{i=1}^n p_i \left|\langle e_i, \bar{\nu} \rangle\right|\right) + j \langle A^T b_1^1, \bar{\nu} \rangle\right],\tag{5.2}$$

where we denote  $A^T b_1^1 = \bar{b}_1^1 \Rightarrow b_1^1 = A \bar{b}_1^1$ . Let the measurement for the one-measurement update be

$$z_1 = Hx_1 + v = HAA^T x_1 + v = \bar{H}\bar{x}_1 + v.$$
(5.3)

Using the coordinate rotation from the outputted conditional mean and error variance, the above characteristic function (5.2) generalizes the results of section 5.3 in [16], where  $\alpha_i = p_i$  and  $z_1$  is replaced by  $\varsigma_1 = z_1 - \bar{H}\bar{b}_1^1 = z_1 - Hb_1^1$ . Therefore, the measurement pdf that generalizes (5.35) in [16] is

$$f_{Z_1} = \frac{1}{\pi} \frac{\sum_{\ell=1}^n p_\ell |h_\ell| + \gamma}{\varsigma_1^2 + \left(\sum_{\ell=1}^n p_\ell |\bar{h}_\ell| + \gamma\right)^2},\tag{5.4}$$

where  $HA = \bar{H} = [\bar{h}_1, \dots, \bar{h}_n]$  from  $\bar{h}_i = \bar{H}e_i$  and by using (5.19e) and (5.19f)  $\sum_{\ell=1}^n \bar{p}_\ell \bar{h}_\ell +$ 

 $\gamma=\bar{H}\bar{p}+\gamma.$  The conditional mean generalizing (5.38) in [16] is

$$\hat{\bar{x}}_1 = \frac{\varsigma_1}{\bar{H}\bar{p} + \gamma}\bar{p} + \bar{b}_1^1 \Rightarrow \hat{x}_1 = A\hat{\bar{x}}_1.$$
(5.5)

Similarly, the rotated conditional variance, where the estimation error is  $\tilde{x}_1 = \bar{x}_1 - \hat{x}_1$ and  $\bar{P} = E[\tilde{x}_1 \tilde{x}_1^T | z_1]$ , is

$$\bar{P} = \left[1 + \frac{\varsigma_1^2}{(\sum_{\ell=1}^n p_\ell |\bar{h}_\ell| + \gamma)^2}\right] \times \begin{bmatrix} \frac{p_1}{|\bar{h}_1|} \left(\sum_{\ell=2}^n p_\ell |\bar{h}_\ell| + \gamma\right) & \dots & -p_1 p_n \operatorname{sgn} \bar{h}_1 \operatorname{sgn} \bar{h}_n \\ \vdots & \vdots & \vdots \\ -p_1 p_n \operatorname{sgn} \bar{h}_1 \operatorname{sgn} \bar{h}_n & \dots & \frac{p_n}{|\bar{h}_n|} \left(\sum_{\ell=1}^{n-1} p_\ell |\bar{h}_\ell| + \gamma\right) \end{bmatrix}.$$
(5.6)

Note that the estimate offset  $\bar{b}_1^1$  does not enter into the rotated conditional variance except in  $\varsigma_1$ . To convert back to the original coordinate frame the conditional variance generalizing (5.42) in [16] is

$$E[\tilde{x}_1 \tilde{x}_1^T | z_1] = A E[\tilde{\tilde{x}}_1 \tilde{\tilde{x}}_1^T | z_1] A^T = A \bar{P} A^T.$$
(5.7)

Using the definition of  $\bar{p}$  in (5.19e) and (5.19f), noting that

 $\sum_{\ell=1}^{n} \bar{p}_{\ell} \bar{h}_{\ell} + \gamma = \bar{H} \bar{p} + \gamma$ , and by adding and subtracting  $\bar{p}_{i} \bar{p}_{i}^{T}$  from the diagonal *ii* elements in (5.6), the rotated variance (5.6) is rewritten using (5.19b), (5.19c), (5.19e) and (5.19f) as

$$\bar{P} = \left[1 + \frac{\varsigma_1^2}{(\bar{H}\bar{p} + \gamma)^2}\right] \left\{ \left(\bar{H}\bar{p} + \gamma\right)\Lambda - \bar{p}\bar{p}^T \right\}.$$
(5.8)

Similarly, the rotated conditional mean from (5.5) is

$$\hat{\bar{x}}_1 = \frac{\varsigma_1 \bar{p}}{\bar{H}\bar{p} + \gamma} + \bar{b}_1^1, \quad \bar{p} = \begin{bmatrix} \bar{p}_1 \\ \vdots \\ \bar{p}_n \end{bmatrix}.$$
(5.9)

Multiply (5.9) on the left by  $\overline{H}$  and some manipulations to produce

$$\bar{H}\hat{\bar{x}}_{1} = \frac{\varsigma_{1}\bar{H}\bar{p}}{\bar{H}\bar{p}+\gamma} + \bar{H}\bar{b}_{1}^{1} \Rightarrow \bar{H}\bar{p} = \frac{\gamma\bar{H}(\hat{\bar{x}}_{1}-\bar{b}_{1}^{1})}{(z_{1}-\bar{H}\hat{\bar{x}}_{1})}$$
$$\Rightarrow \bar{H}\bar{p}+\gamma = \frac{\gamma(z_{1}-\bar{H}\bar{b}_{1}^{1})}{(z_{1}-\bar{H}\hat{\bar{x}}_{1})} = \frac{\gamma\varsigma_{1}}{(z_{1}-\bar{H}\hat{\bar{x}}_{1})}.$$
(5.10)

Turning our attention to reducing the variance equation (5.8) by using (5.10),  $\bar{P}$  becomes

$$\bar{P} = \varsigma_1 \left[ \frac{\gamma^2 + (z_1 - \bar{H}\hat{\bar{x}}_1)^2}{\gamma(z_1 - \bar{H}\hat{\bar{x}}_1)} \right] \Lambda - \left( 1 + \frac{(z_1 - \bar{H}\hat{\bar{x}}_1)^2}{\gamma^2} \right) \bar{p}\bar{p}^T.$$
(5.11)

### Determining $\bar{b}_1^1$ and $\bar{p}$

Important simplifications occur if the  $\bar{P}$  of (5.8) is premultiplied by  $\bar{H}$  and with some manipulations we obtain

$$\bar{H}\bar{P} = \varsigma_1 \left[ \frac{\gamma^2 + (z_1 - \bar{H}\hat{\bar{x}}_1)^2}{\gamma(z_1 - \bar{H}\hat{\bar{x}}_1)} \right] \bar{H}\Lambda - \left( 1 + \frac{(z_1 - \bar{H}\hat{\bar{x}}_1)^2}{\gamma^2} \right) \bar{H}\bar{p}\bar{p}^T = \left\{ \gamma + \frac{(z_1 - \bar{H}\hat{\bar{x}}_1)^2}{\gamma} \right\} \bar{p}^T,$$
(5.12)

and hence

$$\bar{p}^{T} = \frac{\gamma \bar{H} \bar{P}}{\gamma^{2} + (z_{1} - \bar{H} \hat{\bar{x}}_{1})^{2}}.$$
(5.13)

Using (5.10) and (5.13) in (5.9),  $\hat{\bar{x}}_1$  becomes

$$\hat{x}_{1} = \frac{\gamma(\bar{H}\bar{P})^{T}}{(\gamma^{2} + (z_{1} - \bar{H}\hat{x}_{1})^{2})} \frac{\varsigma_{1}}{(\bar{H}\bar{p} + \gamma)} + \bar{b}_{1}^{1}$$

$$= \frac{(\bar{H}\bar{P})^{T}(z_{1} - \bar{H}\hat{x}_{1})}{\gamma^{2} + (z_{1} - \bar{H}\hat{x}_{1})^{2}} + \bar{b}_{1}^{1}$$

$$\Rightarrow \bar{b}_{1}^{1} = \hat{x}_{1} - \frac{(\bar{H}\bar{P})^{T}(z_{1} - \bar{H}\hat{x}_{1})}{\gamma^{2} + (z_{1} - \bar{H}\hat{x}_{1})^{2}}.$$
(5.14)

However, from (5.14),  $b_1^1$  can be written independent of A in terms of the mean  $\hat{x}_1$  and the variance  $P = E[\tilde{x}_1 \tilde{x}_1^T | z_1]$  from (5.7) as

$$A^{T}b_{1}^{1} = \bar{b}_{1}^{1} = A^{T}\hat{x}_{1} - \frac{(z_{1} - HAA^{T}\hat{x}_{1})A^{T}PAA^{T}H^{T}}{\gamma^{2} + (z_{1} - HAA^{T}\hat{x}_{1})^{2}}$$
  

$$\Rightarrow b_{1}^{1} = \hat{x}_{1} - \frac{(z_{1} - H\hat{x}_{1})PH^{T}}{\gamma^{2} + (z_{1} - H\hat{x}_{1})^{2}}.$$
(5.15)

We can now write  $\varsigma_1$  as

$$\varsigma_{1} = z_{1} - \bar{H}\bar{b}_{1}^{1} = z_{1} - Hb_{1}^{1} 
= z_{1} - H\hat{x}_{1} - \frac{(z_{1} - H\hat{x}_{1})HPH^{T}}{\gamma^{2} + (z_{1} - H\hat{x}_{1})^{2}} 
= (z_{1} - H\hat{x}_{1}) \left(\frac{\gamma^{2} + (z_{1} - H\hat{x}_{1})^{2} + HPH^{T}}{\gamma^{2} + (z_{1} - H\hat{x}_{1})^{2}}\right),$$
(5.16)

which is independent of A.

### **Determining** A

Substitution of (5.13) into (5.11) and using (5.19b) and (5.19c) gives

$$A^{T}PA = \theta \Lambda - \frac{A^{T}PH^{T}HPA}{\gamma^{2} + (z_{1} - H\hat{x}_{1})^{2}},$$
(5.17)

which implies (5.18).

### Summary

Based on the results above, these conditions are met if the matrix A satisfies

$$A^T \Psi A = \Lambda, \tag{5.18}$$

where

$$\Psi = P + \frac{PH^T H P}{\gamma^2 + (z - H\hat{x})^2} \in \mathbb{R}^{n \times n},$$
(5.19a)

$$\Lambda = \psi \begin{vmatrix} \frac{\bar{p}_1}{\bar{h}_1} & 0 & \dots & 0 \\ 0 & \frac{\bar{p}_2}{\bar{h}_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \frac{\bar{p}_n}{\bar{p}_n} \end{vmatrix} \in \mathbb{R}^{n \times n}, \ \bar{p}_\ell, \ \bar{h}_\ell \in \mathbb{R},$$
(5.19b)

$$\psi = \frac{\gamma^2 + (z - H\hat{x})^2 + HPH^T}{\gamma} \in \mathbb{R},$$
(5.19c)

$$\bar{p} = \frac{\gamma HPA}{\gamma^2 + (z - H\hat{x})^2} \in \mathbb{R}^n,$$
(5.19d)

$$\bar{H} = HA \in \mathbb{R}^{1 \times n} \tag{5.19e}$$

$$p_{\ell}^{1} = \bar{p}_{\ell} / \text{sgn}(\bar{h}_{\ell}) \in \mathbb{R}_{++}, \quad \ell \in \{1, ..., n\},$$
 (5.19f)

$$b^{1} = \hat{x} - \frac{(z - H\hat{x})PH^{T}}{\gamma^{2} + (z - H\hat{x})^{2}} \in \mathbb{R}^{n},$$
(5.19g)

and  $\gamma$  is the Cauchy pdf modeling parameter for the measurement noise of (2.1). Above,  $\Psi$  is a positive definite matrix, which is a function of the conditional mean and covariance of the (full) neighboring MCE. The important observation is that the left side of (5.18) must be equal to a diagonal matrix. Note that  $\frac{\bar{p}_{\ell}}{h_{\ell}} = \frac{p_{\ell}}{|h_{\ell}|}$ ,  $\ell \in \{1, \dots, n\}$  of (5.19f) are to be strictly positive. Since the left-hand side of (5.18) is a positive definite matrix  $\Psi$  and  $\Lambda$  a positive diagonal matrix, the rotation needed is simply the eigenvectors of  $\Psi$ , which for this positive definite matrix are real and orthogonal. Thus,  $\Lambda$  contains the eigenvalues of  $\Psi$  in diagonal matrix form. To satisfy the constraint of  $AA^T = I$  these eigenvectors are normalized. Note that above  $H \in \mathbb{R}^{1 \times n}$  and when the measurement matrix of (2.1) has more than one row only the most recent measurement (row) is needed for (5.18) and (5.19).

Effectively, (5.18) and (5.19) state that given a desired mean  $\hat{x}$  and covariance P (at the step k + 1), the one-step characteristic function can generate this estimate  $\hat{x}_{k+1}$ ,  $P_{k+1}$ in one step using the measurement  $z_{k+1}$  and measurement noise scaling parameter  $\gamma_{k+1}$  if initialized with the parameters  $A, p^1, b^1$  as formulated above. Note that although the onestep characteristic function can recreate an estimate of a W-step characteristic function, the two characteristic functions will be different.

The construction of the parameter set  $A, p^1, b^1$  described in this section was chosen since the Cauchy conditional covariance is hypothesized to be a tight upper bound to the actual minimum error variance. Numerical evidence in chapter 8 is shown to justify this claim.

### 5.1.2 Software Architecture of the Sliding Window

The MCE algorithm using the sliding window approximation has been implemented in CUDA-C/C++ for distributed evaluation of the characteristic function on GPUs. This is possible since all terms of the MCE are independent and can be computed efficiently in parallel. The sliding window has been implemented as a host-side (CPU) C/C++process that manages a single device-side (GPU) MCE instance. Each MCE is responsible for synchronizing with all other MCE-s as the sensor measurements become available. All MCEs can either share the same underlying GPU or use their respective GPUs, depending upon GPU availability. The data of the windows can furthermore reside on the same compute node or be dispersed across multiple compute nodes and coordinate via socket communication over a local area network. This heterogeneous (CPU/GPU) application design allows for the MCE to be highly extensible to highperformance computing (HPC) clusters, where an application can be dispersed across multiple compute nodes with multiple CPU hosts managing multiple GPU devices. This is useful for the large Monte Carlo experiments of section 8.2. The code is made available at https://github.com/natsnyder1/KingCauchy for academic use. See Fig. 5.1 and source code for further details.

# Chapter 6

# Distributed Computation of the Multivariate Cauchy Estimator

This chapter gives pseudo-codes for the subroutines of the MCE, with attention to their GPU implementations. The mathematical structure of the MCE allows for all its subroutines to be admissible to parallelization; some, albeit, to a larger extent than others. A light overview of GPU terminology is first given to understand the basic nomenclature used under NVIDIA's GPU programming abstraction [24]. In the following sections, the subroutines of the estimator are given in (serial) pseudo-code, followed by comments on how these were parallelized for distributed computation.

### 6.1 The CUDA-C Programming Paradigm

There are two levels of abstraction that comprise a GPU: its hardware level abstraction and its software level abstraction. An overview of the GPU's hardware level abstraction is first given, followed by the software level abstraction with regards to the nomenclature used by NVIDIA's CUDA-C/C++ already programmed interface (API).



 (a) Hardware abstraction of a GPU device.
 Highlighted in red is a single streaming multiprocessor (SM).



(b) Hardware abstraction of a single SM. Each SM has multiple arrays of 32 compute cores (bright green).

Figure 6.1: Hardware abstraction of a GPU

### 6.1.1 Hardware Level Abstraction

GPUs are external 'devices' that a computer program can make use of to execute a task (or portion of a task) in parallel. While GPU device architectures vary, they all contain the same common components arranged into various configurations. Figure 6.1 depicts a general hardware abstraction of a GPU. Each GPU has multiple graphical processing clusters (GPC), each assigned a memory controller. Within each GPC, there are multiple 'streaming multiprocessors' (SM). The SM is the key computational unit of a GPU. Each SM contains arrays of cuda-cores (cores) that are then grouped into arrays of 32. The grouping of 32 cores is called a 'warp' (typically, there are around 4 warps of cores per SM). All cuda cores in a warp simultaneously execute a single instruction per clock cycle, giving the GPU its classification as a 'Single Instruction, Multiple Thread' (SIMT) processor. It is the job of the SM's warp scheduler(s) to manage/schedule the simultaneous execution of its warps (of 32 cores).

Just as a CPU utilizes registers to store local program variables, each SM also contains registers, and divides them amongst its cores to store local program variables. Registers can be thought of as the local program memory available to a core. No other core can view the data held in the registers of another cuda core. All warps of an SM, however, have access to a valuable but small amount of 'shared memory', which the cuda cores can all read and write to. This allows the cores to talk to one another, so to speak. Additionally, all SM's have read/write access to the 'dynamic random access memory' (DRAM) made available on the GPU device via the GPC's memory controller, and is typically referred to as 'global memory' storage. While the amount of local/shared memory is small (kilobytes) and very fast to access, global memory is much more ubiquitous (multiple gigabytes) but slow to access. Accessing data in local memory (registers) takes 1 instruction cycle, data in shared memory several instruction cycles, and data in global memory multiple hundred instruction cycles. The number of cycles varies according to GPU architecture.

Each cuda core is much like a single thread of a CPU. It is able to take an instruction, execute it, and move to the next instruction, abiding by the branching and conditional statements a program may have. The difference between a CPU and a warp of cuda cores is that the warp controller executes a single instruction per warp (32 cores) each clock cycle. For example, if branching logic in a program is hit (say an if-else statement), the warp controller will put to sleep all cores which take the 'else' branch while the if branch executes, and then sleep all cores which take the 'if' branch while the 'else' branch executes. The behavior described is called 'warp divergence' and should be mitigated at all costs. There is no impact, however, if all cores in a warp take the same conditional path. The goal then, is to write GPU programs where all members of a warp execute instructions of the same conditional path.

It should be noted that the GPU (like the CPU) also has both level 1 and level 2 (L1/L2) caches, which aim to minimize memory read times. It may happen that memory access is required from global memory repeatedly, but utilizing shared memory is not optimal for the task. Unlike shared memory, the L1/L2 cache is not controlled by the program but is automatically managed by the device. The L1 cache is faster than L2, but more precious, and uses the leftover shared memory not requested by the cuda cores. This implies memory reads first search the L1 cache, followed by the L2. If neither query is successful, the true global memory read will commence. This is known as a 'cache

miss'.

An extremely important concept is that of coalesced memory accesses versus noncoalesced memory access. Warps read memory in coalesced chunks (cache line), meaning, a chunk of memory (usually 128 or 256 bytes) is accessed by the warp simultaneously. Although global memory reads take many instruction cycles, it is ameliorated somewhat by the fact that if all cores of the warp request data lie within in a coalesced 128-byte chunk, there is no additional overhead to read the memory requested per core. If, however, each core of a warp requests global memory that spans a range of memory addresses larger than 128 bytes, only memory reads within a 128-byte range occur simultaneously. Additional clock cycles are then needed to access the data outside of this region. If all cores request global memory spaced greater than 128 bytes from each other, the time to fulfill the memory read request could be scaled up to a factor of 32. The only saving grace would be L1/L2 cache hits. It becomes imperative then that when accessing global memory, all cores of a warp access coalesced chunks of global memory unless L1/L2 cache hits are highly probable (such as with repeated accesses to memory in a hash table located in global memory). Shared memory does not suffer from the non-coalesced access problem nearly as much, and should be used if non-contiguous memory accesses cannot be avoided (or cache hits are scarce). These concepts explained here can be found in great detail in [24].

#### 6.1.2 Software Level Abstraction

On a software level, programs are constructed and mapped onto device hardware through the thread, block, and grid abstraction. Figure 6.2 depicts this hierarchy. A thread is the most basic of these members. A block is a grouping of threads that holds anywhere from 1 to maximum 1024 threads. A block of threads maps directly to an SM on the GPU, whereas a thread within the block maps to one of the SM's cuda cores. Each thread in a block is given a unique index (0 up to 1023) and sequential groups of 32 threads are designated to be run within the same warp (i.e, thread indices 0-31, 32-63, etc, will be placed in the same warp together). A grid of blocks all contain the same number of



Figure 6.2: Thread block and grid abstraction for mapping compute onto device hardware

threads, and will be evenly distributed amongst the SMs of the GPU. A function written for a GPU is called a kernel, and is 'launched' with a specified number of threads per block (block size) and blocks per grid (grid size).

When a kernel is written for the GPU, it is written such that the thread will carry out the instructions of that kernel. At the software level, it does not matter which thread or block maps to which individual cuda core or SM. The only software-level control is that thread indexes 0-31, 32-63, etc (of the same block) will be placed in the same warp. Defining how many threads are contained within a block allows the SM to partition its resources (such as registers and shared memory) amongst the (many) blocks that have been assigned to it. This is important because threads of the same block are given fastaccess shared memory, through which they can talk/share information/collaboratively use, while threads belonging to different blocks can only 'talk' through global memory. Defining how many blocks you will need for a particular task (i.e., the grid size) allows the device to know how to effectively use its SMs to complete the launched kernel.

It should be noted that while a block can be assigned anywhere from 1 to a maximum of 1024 threads, an individual SM will have less than 1024 cuda cores, typically only 128 (i.e., 4 warps of cuda cores). More to the point, the SM will have more than one block assigned to it as well. This is not a flaw but in fact a design strategy of the SM. The warp scheduler is a master of latency hiding, and achieves this by quickly switching focus from one warp to another the moment an actively running warp needs to write out data to memory, read data from memory, or any operation which involves more than a few cycles of non-compute operation (i.e, latency). The goal of the scheduler is to keep its cuda cores as busy as possible. Typically, the SM wishes to queue up 64 warps (2048 threads) to switch between to best hide latency. However, this is architecture dependent.

Suppose an SM had 128 cuda cores and therefore has 4 warps capable of carrying out compute. Suppose the GPU has 4 SM and a program requests to use 16 blocks, each block consisting of 512 threads. To keep the SM fully busy, each SM will be assigned 4 blocks. This implies that the states of 64 warps (16 warps belonging to each block) will all be given SM resources (shared memory, registers, etc) and actively switched between to minimize overall latency. The 64 warps (2048 threads) that have been assigned device resources are called 'active', as they are actively swapped onto and off the cuda cores with the goal of minimizing latency. The 64 active warps can be at completely different stages of the kernel program, with respect to the others. If the kernel was launched instead using 32 of these blocks, the other 16 blocks would be called 'inactive'. These blocks would be put in a queue, waiting for active warps to complete, until their block is given compute resources and their warps can be designated as 'active'.

Having 64 'active warps' per SM is said to be achieving 100% occupancy of the SM. There are many reasons why 100% occupancy cannot always be achieved. If each thread for a kernel requires more registers than the number of registers (divided by 2048) an SM has, then less than 100% occupancy will be achieved. If several blocks (2048 / block size) assigned to an SM require more shared memory than the SM has to offer, occupancy will also be limited. The goal of the programmer is to write GPU kernels with resource constraints, occupancy, and warp divergence in mind simultaneously, making the programming task much more difficult than programming for a CPU.

# 6.2 An Algorithmic Cookbook for the Distributed Multivariate Cauchy Estimator

### 6.2.1 Overview

The sub-routines of the MCE are now presented in serial pseudo code. After, comments are made on how a GPU kernel was devised to distribute the computation, with respect to both minimizing latency and warp divergence while maximizing occupancy and throughput. The variables of the subroutines follow from the nomenclature given in chapters 2 to 4. For further details on the parallelization, see the source code.

The terms of the MCE are noted with subscript  $i \in [1, ..., N_t^{k|k}]$  at each time step k, and  $m_i^{k|k}$  denotes the number of hyperplanes in the arrangement of term i. It should be noted that for distributed computation on the GPU, it is crucial that a structure of arrays and not an array of structures data format is used to hold the elements  $\{A, p, q, b, c, d, B, \alpha\}$  of all terms. This means all (respective) elements are stored in the same contiguous memory buffer. This enables memory coalescing when warps must access multiple terms.

#### 6.2.2 Time-Propagation

Input:  $A^{k|k}$ ,  $b^{k|k}$ ,  $\Phi_k$ ; Set: *m* to number of hyperplanes in arrangement *i*; for i = 1 to  $N_t^{k|k}$  do  $\begin{vmatrix} // Update$  hyperplane  $A_i^{k|k} \in \mathbb{R}^{m_i^{k|k} \times d}$ ;  $A_i^{k+1|k} \leftarrow A_i^{k|k} \Phi_k^T$ ; // Update offset  $b_i^{k|k} \in \mathbb{R}^{1 \times d}$ ;  $b_i^{k+1|k} \leftarrow b_i^{k|k} \Phi_k^T$ ; end Return:  $A_i^{k+1|k}$ ,  $b^{k+1|k}$ ;

Algorithm 1: Time Propagation Algorithm

Note that the time propagation routine here does not append  $\Gamma_k^T$  and  $\beta_k$  to the  $A_i^{k|k}$ and  $p_i^{k|k}$  terms, respectively. This is because it would corrupt the memory storage of all terms > *i*, which are stored contiguously with term *i* in the memory buffer. Furthermore, appending these elements would then require an expensive global memory write operation to a new (and non-L1/L2-cached) global memory buffer. Instead, these elements are dealt with next in time-propagation coalignment, pending the rows of  $\Gamma_k^T$  are not coalign with any rows of  $A_i^{k|k}$ . Therefore, the GPU program simply needs to do two large matrix multiplies:  $A^{k+1|k} = A^{k|k} \Phi_k^T$  and  $b^{k+1|k} = b^{k|k} \Phi_k^T$  where  $A^{k|k} \in \mathbb{R}^{N_e^{k|k} \times d}$  is the full memory buffer of shape,  $N_e^{k|k} = \sum_{i=1}^{N_t^{k|k}} m_i^{k|k}$ , and  $b^{k|k} \in \mathbb{R}^{N_t^{k|k} \times d}$ , which is ideal GPU work.

### 6.2.3 Time-Propagation Co-alignment

// We complete the following algorithm over all  $i \in [1, 2, ..., N_t^{k|k}]$ ; **Input:**  $A_i^{k+1|k}$ ,  $p_i^{k|k}$ ,  $\Gamma_k$ ,  $\beta_k$  // Using  $A_i^{k+1|k}$  of algorithm 1; **Set:**  $\epsilon \in \mathbf{R}_+$  with epsilon a small positive number; Set:  $\mathscr{F} \in \mathbf{B}^r$  to all True, where  $r = \operatorname{cols}(\Gamma_k)$ ; Set:  $p_i^{k+1|k} \leftarrow p_i^{k|k} \circ \operatorname{norm}(A_i^{k+1|k}, \operatorname{axis}=1) // \operatorname{row} \text{ wise L2 norm};$ **Normalize:**  $A_i^{k+1|k}$  // row wise L2 normalization; Set:  $\beta_k \leftarrow \beta_k \circ \operatorname{norm}(\Gamma_k, \operatorname{axis}=0) // \operatorname{column}$  wise L2 norm; **Normalize:**  $\Gamma_k$  // column wise L2 normalization; for j = 1 to r do // Testing for coalignment with  $\Gamma_k[:, j]$ . **1** is a one vector.;  $\gamma_j = \Gamma_k[:,j] // \gamma_j \in \mathbf{R}^d$ ;  $indx = \operatorname{argwhere}\left(\left(1 - |A_i^{k+1|k}\gamma_j|\right) < \epsilon\right) // \text{ row indices of coalignment;}$  $assert(len(indx) \leq 1) // Only$  one HP can coalign, else there is a bug; if  $len(indx) \neq 0$  then  $\begin{array}{l} // \text{ Combine corresponding elements of } p_i^{k|k} \text{ and } \beta_k; \\ p_i^{k+1|k}[indx] \leftarrow p_i^{k+1|k}[indx] + \beta_k[j]; \\ \mathscr{F}[j] = \text{False}; \end{array}$ else | // hyperplane j does not coalign, it is unique, no action end end

**Append:**  $\Gamma_k^T$  to  $A_i^{k+1|k}$  for indices of  $\mathscr{F}[j] ==$  True; **Append:**  $\beta_k$  to  $p_i^{k+1|k}$  for indices of  $\mathscr{F}[j] ==$  True; **Return:**  $A_i^{k+1|k}$ ,  $p_i^{k+1|k}$ ; **Algorithm 2:** Time Propagation Coalignment Algorithm

Here the time propagation coalignment (TPC) algorithm constructs the 'final'  $A_i^{k+1|k}$ and  $p_i^{k+1|k}$ , taking into account any coalignment which may occur between rows of  $\Gamma_k^T$  and rows of  $A_i^{k+1|k}$ . The TPC algorithm for the GPU allocates a 2-D array of  $(d \times \max(m_i^{k|k}, i \in [1, ..., N_t^{k|k}])$  threads per block, using each row of threads per block to check whether a row of the hyperplane arrangement coaligns with a column of  $\Gamma_k$ , simultaneously. Each block allocates shared memory space for  $A_i^{k+1|k}, p_i^{k+1|k}, \Gamma_k$ , and  $\beta_k$ , first reading these values in from global memory and conducting all operations locally (and fast) on the SM itself.

The trouble now, is that the number of hyperplanes a term has after coaligning is not a priori known (as well as the execution order of the blocks on SMs). Since the blocks must return the terms to their respective (contiguous) memory buffers, they will need to coordinate with one another to avoid overwriting each other's data when writing results back out to global memory. This problem is solved by using a set of global integer counters, where each block uses an 'atomic add' <sup>1</sup> operation on a counter to find its offset in the global memory buffers. The number of counters spans the possible number of hyperplanes, i.e., one counter per possible arrangement size after coalignment, with each counter initially set to zero. Only one thread per block will access a respective counter, while the other threads in the block carry out no work until the counter is incremented<sup>2</sup>. The result, is the blocks can return the terms into coalesced memory buffers. Specifically, all terms with the same hyperplane shape are placed contiguously in one memory buffer, and all terms of another hyperplane shape are in their own contiguous memory buffer. Therefore, the TPC algorithm returns a coalesced array of arrays for all terms, where the final counts of the counter variables indicate the number of terms (size of the array) with arrangements of a certain size. This strategy is used as well in measurement update coalignment.

<sup>&</sup>lt;sup>1</sup>An atomic operation makes certain only one thread (of all threads in all allocated blocks) conducts read/write operations on a memory address at a time. This 'serializes' read/write operations to the memory address in question, preventing race conditions between multiple threads.

 $<sup>^2{\</sup>rm This}$  is not as bad as it appears, as the counters are accessed so often, they will be placed into L1 or L2 cache.

### 6.2.4 Measurement Update (Child-Term Generation)

// We complete the following algorithm over all  $i \in [1, 2, ..., N_t^{k+1|k}]$  (Note  $N_t^{k+1|k} = N_t^{k|k}$ );

// Note:  $m_i^{k|k}$  is the size of the *i*-th hyperplane arrangement before time-prop (algorithm 1);

 $\begin{array}{l} \text{Input: } A_{i}^{k+1|k}, \, p_{i}^{k+1|k}, \, b_{i}^{k+1|k}, \, H_{k+1}, \gamma_{k+1}, m_{i}^{k|k}, \alpha_{i}^{k-1|k-1}; \\ \text{Set: } k \leftarrow k+1; \\ \text{Set: } \hat{\lambda}_{i}^{k|k-1} = \text{sgn} \left( A_{i}^{k|k-1} [1:m_{i}^{k-1|k-1},:]H_{k}^{T} \right) \in \{\pm 1\}^{m_{i}^{k-1|k-1}}; \\ // \text{ Equation (3.8) returns children ;} \\ // \{A_{t}^{k|k}, p_{t}^{k|k}, b_{t}^{k|k}, c_{t}^{k|k}, d_{t}^{k|k} \}, \quad t \in [1, ..., m_{i}^{k|k-1} + 1]; \\ \text{Call: Equation (3.8), Inputs: } z_{k}, A_{t}^{k|k-1}, p_{t}^{k|k-1}, b_{t}^{k|k-1} \, // \text{ Note } m_{i}^{k|k} = m_{i}^{k|k-1}; \\ // \text{ All child terms } t \text{ hold a pointer to the updated parent vector } \alpha_{i}^{k|k-1}; \\ \text{Set: } \alpha_{i}^{k|k-1} \leftarrow \alpha_{i}^{k-1|k-1} \circ S\left(\hat{\lambda}_{i}^{k|k}, \text{ for } t \in [1, ..., m_{i}^{k|k-1} + 1] \text{ and } \alpha_{i}^{k|k-1}; \\ \text{ Algorithm 3: Measurement Update Algorithm} \end{array}$ 

Equation (3.8) is the main driver of the child generation process, which is why algorithm 3 simply calls it. As the terms are now stored in an 'array of arrays' from algorithm 2, the GPU MU kernel is launched on each array of different-sized arrangements sequentially. The terms of all children of a particular (parent term's) arrangement size are generated per kernel launch. Let  $N_t(m^{k|k-1})$  be the number of terms with m hyperplanes in its arrangement at k|k-1 (which is the value stored in the counters used by algorithm 2. Each kernel launch for the MU algorithm uses a grid of  $N_t(m^{k|k-1})$  blocks, each allocated with a 2-D thread array of size  $(m^{k|k-1} + 1 \times m^{k|k-1} + 1)$ . Each row of threads then simultaneously constructs one child term, and therefore the block generates the parent's  $m_i^{k|k-1} + 1$  child terms. Note that not all threads in a block are used to construct elements  $p_t^{k|k}, b_t^{k|k}, c_t^{k|k}, d_t^{k|k}$ , however, these elements can be generated quickly and few compute cycles are wasted.

The child terms generated by each block are placed back into a contiguous structure of arrays memory buffer, one array per element A, p, b, c, d. This is so the measurement update coalignment algorithm that follows can re-place these terms back into the 'array of arrays' data structure after coalignment has been checked. Therefore, we see the coalignment routines algorithms 2 and 5 take as input a 'structure of arrays' data structure, and output a 'structure of array of arrays' data structure, with each array holding the elements of all terms of a given hyperplane size, post-coalignment.

Regardless of the  $\alpha$ -generation technique chosen, the parent alpha vector  $\alpha_i^{k-1|k-1}$  must be updated. Since the operation is a Hadamard product, all threads can loop over the coefficients of  $\alpha_{il}^{k-1|k-1}$  until all coefficients have been updated with the corresponding element of  $S\left(\hat{\lambda}_i^{k|k-1}\right)_l$ ,  $l \in [1, ..., s_g(m_i^{k-1|k-1}, d)]$ . Each child term must maintain a pointer to the updated vector, in order to evaluate its g-value next. The children, lastly, are continuously re-indexed as  $i \in [1, 2, ..., N_t^{k|k}]$ , however, the index t of each child term must be remembered, as it is needed in algorithms 4 and 8 to evaluate the g-function (3.20).

### 6.2.5 G Evaluation

// We complete the following algorithm over all  $i \in [1, 2, ..., N_t^{k|k}]$ ;

// Note: t is the child's index from algorithm 3;

// Note:  $m_i^{k-1|k-1}$  is the size of the child's parent hyperplane arrangement before algorithm 1;

$$\begin{split} & \text{Input: } A_{i}^{k|k}, \, p_{i}^{k|k}, \, b_{i}^{k|k}, \, c_{i}^{k|k}, \, d_{i}^{k|k}, \, \bar{\nu}, \, t, m_{i}^{k-1|k-1}, \, \alpha_{i}^{k|k-1}; \\ & \text{Set: } \lambda_{i}^{k|k} = \text{sgn} \left( A_{i}^{k|k} \bar{\nu} \right) \in \{\pm 1\}^{m_{i}^{k|k}}; \\ & \text{Set: } \lambda^{+} = \left[ \lambda_{i}^{k|k} [1:t-1], \, 1, \, \lambda_{i}^{k|k} [t:m_{i}^{k-1|k-1}] \right]^{T} \in \{\pm 1\}^{m_{i}^{k-1|k-1}}; \\ & \text{Set: } \lambda^{-} = \left[ \lambda_{i}^{k|k} [1:t-1], \, -1, \, \lambda_{i}^{k|k} [t:m_{i}^{k-1|k-1}] \right]^{T} \in \{\pm 1\}^{m_{i}^{k-1|k-1}}; \\ & \text{Set: } y_{gi}^{k|k} = \left( p_{i}^{k|k} \right)^{T} \lambda_{i}^{k|k}; \\ & \text{Evaluate: } g_{i}^{k|k} (\bar{\nu}) = \frac{1}{2\pi} \left[ \frac{S(\lambda^{+})^{T} \alpha_{i}^{k|k-1}}{jc_{i}^{k|k+1} + y_{gi}^{k|k}} - \frac{S(\lambda^{-})^{T} \alpha_{i}^{k|k-1}}{jc_{i}^{k|k-1} + y_{gi}^{k|k}} \right]; \\ & // \text{ Calling (2.6d) returns } \bar{y}_{ei}^{k|k} = y_{ei}^{k|k} (\bar{\nu}); \\ & \text{Call: (2.6d), Inputs: } A_{i}^{k|k}, p_{i}^{k|k}, b_{i}^{k|k}, \bar{\nu}; \\ & \text{Return: } g_{i}^{k|k} (\bar{\nu}), \, \bar{y}_{ei}^{k|k}; \\ & \text{ Algorithm 4: G-Evaluation Algorithm } \end{split}$$

Here, a point in space  $\bar{\nu} \in \mathbb{R}^d$  is chosen, and used to evaluate each child term's  $g_i^{k|k}(\nu)$ ,  $y_{ei}^{k|k}(\nu)$  functions (i.e, (2.6d) and (3.20)), which are then used to compute the moments by (2.8) and (2.9). As (3.20) is a complicated expression, it has been spelled out above. The GPU uses a single warp (32-threads) per block, with a grid size of  $N_t^{k|k}$ . Each block evaluates (2.6d) and (3.20) for a single term. Evaluating  $\lambda^{\pm}$  does cause slight warp divergence (and many threads in the warp are unused), but the main expense per kernel call is evaluating the (complex-typed) inner product required to compute the numerators of  $g_i^{k|k}(\nu)$ . Because the data type of  $\alpha_i^{k|k-1}$  is a double complex-typed array,

each element is size 16 bytes. Therefore, reading 32 elements of  $\alpha_i^{k|k-1}$  does not fit in the warp's cache line when reading from global memory.

To get around this, 8 elements of the  $\alpha_i^{k|k-1}$  vector are read at a time. The first 8 threads sum the real part of the left-hand numerator, the next 8 threads sum the complex part of the left-hand numerator, the next 8 threads sum the real part of the right-hand numerator, and the last 8 threads sum the complex part of the right-hand numerator. Doing so, the entire warp is kept busy and all global memory reads are now cache aligned, as 16 \* 8 = 128 bytes. Next, d threads are used to compute  $\bar{y}_{ei}^{k|k}$ , while only one thread can finish the evaluation of  $g_i^{k|k}(\bar{\nu})$ , while the other threads wait. Lastly, all  $g_i^{k|k}(\bar{\nu})$ ,  $\bar{y}_{ei}^{k|k}$  are written out to a contiguous memory buffer, which is used to compute the moments by evaluating (2.8) and (2.9) via a parallel summation operation (see source code). Note that it is not necessary to coalign before evaluating the moments and the g-values.

### 6.2.6 Measurement Update Co-alignment

// We complete the following algorithm over all  $i \in [1, 2, ..., N_t^{k|k}]$ ; Input:  $A_i^{k|k}, p_i^{k|k};$ Set:  $\epsilon \in \mathbf{R}_+$  // with epsilon a small positive number; Set:  $p_i^{k|k} \leftarrow p_i^{k|k} \circ \operatorname{norm}(A_i^{k|k}, \operatorname{axis}=1)$  // row wise L2 normalization; **Normalize:**  $A_i^{k|k}$  // row wise L2 normalization; Set:  $q_i^{k|k} \leftarrow p_i^{k|k} /$  copy operation; Set:  $\mathscr{F} \in \mathbf{B}^{m_i^{k|k}}$  to all True, and variable  $\ell = 1$ ; **Declare:**  $\operatorname{map}_{i}^{k|k} = \mathbf{0}^{m_{i}^{k|k}}$ ,  $\operatorname{sign_map}_{i}^{k|k} = \mathbf{0}^{m_{i}^{k|k}} / / m_{i}^{k|k}$  length vectors; for j = 1 to  $m_i^{k|k} - 1$  do if  $\mathscr{F}[j]$  then // Test hyperplane j for coalignment;  $a_{r} = A_{i}^{k|k}[j,:] // a_{r} \in \mathbf{R}^{d};$   $A_{c} = A_{i}^{k|k}[j+1:m_{i}^{k|k},:] // A_{c} \in \mathbf{R}^{\left((m_{i}^{k|k}-j)\times d\right)};$ indxs = argwhere( $(\mathbf{1} - |A_{c}a_{r}|) < \epsilon$ ) // array of indexes  $\in [1, ..., m_{i}^{k|k} - j];$ map $_{i}^{k|k}[j] = \ell;$  $\operatorname{sign}_{k|k}[j] = 1;$ if  $len(indxs) \neq 0$  then for *indx* in *indxs* do  $\mathscr{F}[indx+j] = False;$ 
$$\begin{split} & \max_{i}^{k|k} [indx+j] = j; \\ & p_{i}[j] = p_{i}[j] + p_{i}[indx+j]; \\ & // \text{ Find sign difference between hyperplane's j and indx;} \\ & s = \text{sign_difference}(a_{r}, A_{c}[indx, :]) // \text{ either } (+1/\text{-}1); \end{split}$$
 $q_i[j] = q_i[j] + s * q_i[indx + j];$ sign\_map\_i^{k|k}[indx + j] = s; end else | // hyperplane j does not coalign, it is unique, no action end  $\ell \leftarrow \ell + 1$ | // hyperplane j has already been coaligned, no action end end

**Return:**  $A_i^{k|k}, p_i^{k|k}, q_i^{k|k}$ , for True indexes of  $\mathscr{F}$ , and  $\operatorname{map}_i^{k|k}$ , sign\_map $_i^{k|k}$ ; **Algorithm 5:** Measurement Update Coalignment Algorithm

The measurement update coalignment algorithm is similar to algorithm 2, with the exception that now all hyperplanes are tested for coalignment with respect to each other (and not just with  $\Gamma_k$ ). The key difference here is that the arrays 'map' and 'sign\_map' are vitally important, and need to be constructed. These will be used to construct the

sign vectors  $\lambda^{\pm}$  in algorithms 8 and 11 'post coalignment'.

Therefore, to properly construct these arrays, the top-level for loop in algorithm 5 cannot be unrolled in the GPU implementation, unfortunately.  $N_t^{k|k}$  blocks are launched each with one warp. The warp can evaluate the 'argwhere' statement in parallel but is only able to utilize a few threads to carry out the instructions following it. Luckily, the inputs of algorithm 5 are first read into shared memory, mitigating the latency of read operations and making the low warp utilization a small issue. Similar to algorithm 2, global counters are used to write out the sorted 'array of arrays' data structure of all terms after coalignment is complete. This GPU algorithm, overall, is still exceptionally quick.

### 6.2.7 Alpha Parameterization using Incremental Enumeration

There are two strategies presented to evaluate the parameters  $\alpha_i^{k|k}$ , one using cell enumeration, and the other using Pinchasi's method. In this section, the pseudo-code is given to properly evaluate the (complicated) expression  $\alpha_i^{k|k} = S(B_i^{k|k})^{\dagger} \bar{g}_i^{k|k}$ . The first step is determining the enumeration matrices  $B_i^{k|k}$  for all terms. Next,  $\bar{g}_i^{k|k}$  of (3.36) must be constructed. The enumeration matrix must then be row-wise expanded as  $S(B_i^{k|k})$ , followed by a routine to solve the system of equations to recover  $\alpha_i^{k|k}$ .

#### **Incremental Enumeration**

The calling function for Inc-Enu can be given as

// We complete the following algorithm over all  $i \in [1, 2, ..., N_t^{k|k}]$ ; Input:  $A_i^{k|k}, \bar{\nu}$ ; Flip:  $A_{il}^{k|k} \leftarrow -A_{il}^{k|k}$  if  $A_{il}^{k|k} \bar{\nu} < 0$ ,  $\forall l \in [1, ..., m_i^{k|k}]$  // Store these flips; Set:  $B_i^{k|k} = \{\}$  // Set that will hold sign vectors of the cells of  $A_i^{k|k}$ ; Set:  $s = \{1\}$  // Sign-sequence to be tested (validated/invalidated); Call: inc\_enu( $A_i^{k|k}, \bar{\nu}, B_i^{k|k}, s$ ) // Fills  $B_i^{k|k}$  with a sign-vector to identify each cell of  $A_i^{k|k}$ ; Set:  $B_i^{k|k} \leftarrow B_i^{k|k} \bigcup -B_i^{k|k}$  // Append the negative of  $B_i^{k|k}$  to the set; Flip: Sign-elements of  $B_i^{k|k}$  corresponding to the flipped hyperplanes  $A_{il}^{k|k}$ ; Unflip:  $A_i^{k|k}$  // Corresponding to the flipped hyperplanes  $A_{il}^{k|k}$ ; Return:  $B_i^{k|k}$ ; Algorithm 6: Incremental Enumeration Calling Algorithm The calling function uses a recursive depth first search (driver function) that can be

written as

Input: A, x, B, s // x is an interior cell point. To begin, it is  $\bar{\nu}$ ; Set: l = len(s); Set: m = rows(A); if l == m then  $| B \leftarrow B \bigcup s$ ; Return:  $\emptyset$ ; end

// is\_cell function tests if s is valid by solving an LP. Updates x with LP solution if s is valid;

if  $\langle A[l+1,:], x \rangle > 0$  then

 $\begin{vmatrix} s \leftarrow s \bigcup 1; \\ \text{inc\_enu}(A, x, B, s) \ // \ \text{s is restored to length } l+1 \ \text{after returning from the call}; \\ s[l+1] = -1 \ // \ \text{Now validate/invalidate the opposite;} \\ \textbf{if } is\_cell(s, A[1:l+1,:], x) \ \textbf{then} \\ | \ \text{inc\_enu}(A, x, B, s) \ // \ \text{same applies here;} \\ \textbf{end} \end{vmatrix}$ 

else

end  $s \leftarrow s \bigcup -1;$ inc\_enu(A, x, B, s) // s is restored to length l + 1 after returning from the call; s[l+1] = 1 // Now validate/invalidate the opposite; if  $is\_cell(s, A[1:l+1,:], x)$  then | inc\_enu(A, x, B, s) // same applies here; end Return:  $\emptyset$ ;

Algorithm 7: inc\_enu (driver function)

The (serial) Inc-Enu algorithm uses a straightforward recursive implementation. See section 3.2 for details on the Phase-I LP solved within the 'is\_cell' routine, and further details on Inc-Enu. The GPU implementation, while less memory efficient, takes advantage of the massive parallelization possible by recasting the depth-first search to a breadth-first search. As the proposed strategy uses a combination of sequential and parallel computing operations, the method is coined 'Hybrid Inc-Enu' (HIE).

While Inc-Enu conducts a depth-first search, HIE is a parallel breadth-first search reformulation that operates on a forest (many trees) of enumeration tasks, solving a grid of computation at each sequential step. Figure 6.3 illustrates this parallel enumeration process. To begin, HIE is given all N arrangements of m-hyperplanes to be enumerated. HIE takes the results of the first grid at step 1 (denoted S1) as input to solve the next grid, S2. Note that all enumeration tasks located within a given step (or depth) of the forest are independent of one another and can be solved in parallel. We see that at every step of parallel computation, CUDA blocks are assigned to the active enumeration tasks of each tree. Each block of threads solves an LP to validate/invalidate the proposed signsequence using the simplex method [27] with tableau size  $((SX + 2) \times (2d + 2))$ , at each step S'X'. See [27] and section 3.2 for reassurance that the block size is indeed the size of the resulting simplex tableau for the LP. Each block produces a left and right child (also CUDA blocks), if and only if the block's proposed sign-sequence is found valid. The left child (seen in green) is provided with both the sign-sequence and computed feasible point of its parent, and simply needs to append which halfspace the known feasible point lies in with respect to the next hyperplane to be enumerated. Left children are computationally cheap, as these blocks do not need to solve an LP. Each right-child block (seen in gray/red) is given the validated parent sign sequence concatenated to the *opposite* of the sign computed by the left child, as input. The right child is responsible then for solving a feasibility LP to validate this proposed sign sequence.



Hybrid Incremental Enumeration (HIE)

Figure 6.3: Hybrid Inc-Enu example forest, with N arrangements of *m*-hyperplanes in dimention *d*. Blocks (thread blocks) within the grid at each step are solved in parallel. Red/Black boxes indicate a CUDA block solved an LP. Green boxes indicate no LP was needed to validate the sign sequence.

Programmatically, this translates to developing a parallel Phase-I simplex method [26], where each CUDA block is responsible for computing a simplex tableau (LP) of a single, proposed sign sequence. Previous works on GPU linear programming [32] focus their attention on using the GPU-compute resources to parallelize tableau operations for individual large and/or sparse LPs. Instead, we focus the GPU-compute resources on efficiently parallelizing many *small* Phase-I LPs for throughput. The authors in [33] propose a Phase I and Phase II simplex method for simultaneously solving small batched LPs. Here, we develop HIE to streamline the inputs and outputs of the batched Phase-I simplex algorithm proposed in [33] explicitly for solving batched cell enumerations.

We note that HIE sequentially runs m-1 parallel grid computations (GPU kernel launches) to fully enumerate the cells of all N arrangements. As seen in Figure 6.3, the number of needed enumerations grows at each parallel computation step. As HIE descends further down into levels of the forest, note that for arrangements larger than 4 hyperplanes (*e.g.*, 20, 30), the forest becomes immensely sparse in relation to the possible number of permutations  $\{\pm 1\}^m$ . See source code for details on simplex tableau parallelization.

#### **G-Evaluation** Per Cell

After HIE completes, the sign vectors (rows) of  $B_i^{k|k}$  are used to construct  $\bar{g}_i^{k|k}$ , which is (3.20) evaluated for each cell of a term's arrangement  $A_i^{k|k}$ . The challenge here is to correctly construct the  $\lambda^{\pm}$  vectors after the coalignment step, which are used to evaluate the numerators of (3.20) for each cell. // We complete the following algorithm over all  $i \in [1, 2, ..., N_t^{k|k}]$ ; **Input:**  $B_i^{k|k}, q_i^{k|k}, c_i^{k|k}, d_i^{k|k}, \operatorname{map}_i^{k|k}, \operatorname{sign\_map}_i^{k|k}, \alpha_i^{k|k-1}, t, m_i^{k-1|k-1}$ . ; **Set:**  $c_c(A_i^{k|k}) = \operatorname{rows}(B_i^{k|k}) / /$  Number of cells in arrangement  $A_i^{k|k}$ ; Set:  $\bar{g}_i^{k|k} = \mathbf{0}^{c_c(A_i^{k|k})};$ // Iterate over all sign-vectors  $\bar{\lambda} \in B_i^{k|k}$ ; for b = 1 to  $c_c(A_i^{k|k})$  do b = 1 to  $c_c(A_i^{m,n})$  do  $\bar{\lambda} = B_i^{k|k}[b,:] // \bar{\lambda} \in \{\pm 1\}^{m_i^{k|k}};$ 
$$\begin{split} y_{gi}^{k|k} &= \left(q_i^{k|k}\right)^T \bar{\lambda}; \\ \mathbf{Set:} \quad \lambda^+ &= \mathbf{0}^{m_i^{k|k}}, \ \lambda^- &= \mathbf{0}^{m_i^{k|k}}, \ \ell = 1; \\ // \ t \text{ is the child index, and } m_i^{k-1|k-1} \text{ is the size of the parent arrangement (see algorithm 3);} \end{split}$$
for j = 1 to  $m_i^{k-1|k-1}$  do  $\begin{array}{c|c} \text{ or } j = 1 \ to \ m_i^{\kappa^{-1}|\kappa^{-1}} \ \mathbf{do} \\ \hline \mathbf{if} \ j \neq t \ \mathbf{then} \\ & s = \text{sign\_map}_i^{k|k}[\ell]; \\ idx = \text{map}_i^{k|k}[\ell]; \\ \lambda^+[j] = s * \bar{\lambda}[idx] ; \\ \lambda^-[j] = s * \bar{\lambda}[idx] ; \\ \ell \leftarrow \ell + 1; \\ \hline \mathbf{else} \\ & \lambda^+[j] = 1 ; \\ \lambda^-[j] = -1 ; \\ \mathbf{end} \end{array}$ end **Evaluate:**  $\bar{g}_i^{k|k}[b] = \frac{1}{2\pi} \left[ \frac{S(\lambda^+)^T \alpha_i^{k|k-1}}{jc_i^{k|k} + d_i^{k|k} + y_{ai}^{k|k}} - \frac{S(\lambda^-)^T \alpha_i^{k|k-1}}{jc_i^{k|k} - d_i^{k|k} + y_{ai}^{k|k}} \right];$ end **Return:**  $\bar{g}_i^{k|k}$ ; **Algorithm 8:** G-Evaluation Per Cell Algorithm

Similar to algorithm 4, the main computational expense is evaluating (3.20) in each cell of a hyperplane arrangement of term i. Here, however, both for loops of algorithm 8 can be unrolled. Each block is composed of a  $16 \times 16$  grid of threads. Kernel launches are of  $N_t(m^{k|k})$  blocks (see algorithm 3), computing the gs for all terms of a given arrangement size in each kernel launch. Each row of the thread block works on evaluating (3.20) in a particular cell, allowing 16 cells to be evaluated simultaneously, before looping over another 16 (and so on). Warp divergence is encountered when building  $\lambda^{\pm}$  in shared memory, however latency is low as  $\operatorname{map}_{i}^{k|k}$  and  $\operatorname{sign}_{i} \operatorname{map}_{i}^{k|k}$  are read into shared memory before building  $\lambda^{\pm}$  (also in shared memory). The GPU strategy for evaluating the numerators of (3.20) is similar to algorithm 4, except now 4/16 threads are used for
the real and complex parts of the left and right-hand side numerators, respectively.

#### Basis Matrix and Solving for Alpha

Creating the basis matrix  $S(B_i^{k|k})$  relies on a special data structure that provides the indices of products of the sign vectors (see section 3.3) to construct its combinatorial sequence. The algorithm is not given, as this part is considered straight forwards, given an appropriate data structure for the task is first created. However, it should be noted that constructing each element of  $S(B_i^{k|k})$  is independent of constructing the other elements. This allows 1 thread to work on this element without coordination of the other elements. Therefore, no warp divergence is encountered. Furthermore, each row of  $B_i^{k|k}$  is expanded in parallel, allowing full warp utilization at each instruction cycle. A block size of  $16 \times 16$  or  $16 \times 32$  is appropriate for the task of constructing the basis matrix from the enumeration matrix.

Solving the system  $S(B_i^{k|k})\alpha_i^{k|k} = \bar{g}_i^{k|k}$  can use any one of the numerous solvers for large linear systems of equations. In a CPU implementation, a LAPACK [34] routine such as 'dgels' could be considered, which relies on QR matrix-factorization. For the GPU, batched versions of dgels or PLU decomposition can alternatively be considered from the 'CUBLAS' [35] library. However, these (factor-then-solve) methods were seen to be sub-optimal in this application. The time-savings of factorization methods like PLU or QR stem from their ability to solve multiple right-hand sides using a single factorization. Here, we only have one right-hand side (i.e.,  $\bar{g}_i^{k|k}$ ) per factorization. Moreover, due to cell degeneracy in  $B_i^{k|k}$ , the number of rows in  $S(B_i^{k|k})$  are not the same amongst different terms *i*. Furthermore, the typing between  $S(B_i^{k|k})$  and  $\bar{g}_i^{k|k}$  is real and complex, respectively. These caveats make finding an off-the-shelf implementation of (already limited) GPU solvers difficult.

A home-brewed GPU Gaussian Elimination solver was seen to have an impressive computation time and sufficient accuracy when solving batches of these systems on the GPU, greatly outperforming the other methods enumerated above. See [36] for a CPU implementation of Gaussian Elimination. This algorithm is not a part of the CUBLAS library, currently, and could be considered a contribution to the software deck. Much of the computation of GPU Gaussian Elimination is susceptible to parallelization. For example, when a pivot is chosen, all rows (or columns) beneath (right of) the pivot can be updated simultaneously. Choosing a pivot, however, can take many instruction cycles (as well as row/column swap operations that follow), during which all but one thread will have to be put asleep. See the source code for the GPU implementation.

# 6.2.8 Alpha Parameterization using Pinchasi's Method

Chapter 4 gives a high-level explanation of the steps required to construct an algorithm based on Pinchasi's insights [23]. For completeness, select pseudo algorithms of sections 4.2.1 to 4.2.7 are given in this section, as well as the full pseudo code to carry out the routine. The intent of including these pseudo codes is twofold. First, it is the hope that these pseudo algorithms help unveil some of the subtler steps required from sections 4.2.1 to 4.2.7. Second, it is the hope that when newer and more powerful parallel processors become a mature technology, such as the intelligent processing unit, or IPU [37], this section would be of use to programmers looking to implement this method of parameterization for (likely to be, orders of magnitude) faster implementations of the MCE. The following pseudo algorithms drop the superscript k|k and term index i, for brevity. These routines, however, are called for all terms i of the MCE and at each step k|k.

### Upper Cell of a Vertex

**Input:** Indices I with |I| = d, Vertex  $x_I$ , Hyperplanes  $A_I$ ; **Set:** Upper region of vertex point  $p_{x_I} \leftarrow x_I$ ; for i = 1 to d do Form new  $I_{d-1} \subset I$  and  $|I_{d-1}| = d-1$ ; Form  $A_{I_{d-1}} \in \mathbb{R}^{d-1 \times d} \subset A_I$  with  $I_{d-1}$ ; // Constructing the *d*-dimensional cross product below; Declare memory for  $p_i \in \mathbb{R}^d$ ;  $I_{d^{-1}} \leftarrow \operatorname{sort}(I_{d^{-1}}) / / \operatorname{Ascending Order};$ for j = 1 to d do Form  $I'_{d-1} = I_{d-1} \setminus I_{d-1}[j];$ // Forming the matrix minor  $A'_{I_{d-1}} \in \mathbb{R}^{d-1 \times d-1};$   $A'_{I_{d-1}} \leftarrow \text{Take columns } I'_{d-1} \text{ out of } A_{I_{d-1}};$   $p_i[j] = (-1)^{j+1} \det(A'_{I_{d-1}});$ end if  $p_i[d] < 0$  then  $| p_{x_I} = p_{x_I} - p_i$ else  $| \quad p_{x_I} = p_{x_I} + p_i$ end end Return:  $p_{x_I};$ Algorithm 9: Upper Cell of a Vertex (see section 4.2.2)

Unpriming the Coefficients

Input: Sets  $I, I', \alpha'_I$  and  $\alpha \in \mathbb{R}^{s_g(m,n)}$ ; Set:  $\overline{I} = I \setminus I'$ ; for  $I'' \subset I'$  do | Form  $I_p = \overline{I} \bigcup I''$ ;  $\alpha_{I_p} = \alpha_{I_p} + (-1)^{|I''|} \alpha'_I$ ; end Return:  $\alpha$ ; Algorithm 10: Unpriming the Coefficient for  $\alpha'_I$  (see section 4.2.4)

#### Full Algorithm

Using algorithms 9 and 10, the full routine can be written as

Input:  $A \in \mathbb{R}^{m \times n}$ ,  $\alpha \in \mathbb{R}^{s_g(m,n)}$ ; **Declare and Perturb:**  $b \in \mathbb{R}^m$  // Small random affine offset; **Rotate:** A, s.t no hyperplane is coaligned with a unit direction  $e_i, i \in [1, ..., d]$ ; Set:  $\alpha = \mathbf{0}^{s_g(m,n)}, x_F \in \mathbb{R}^n = \mathbf{0}^n, I_m = \{1, .., m\};$ for d = n to 1 do // Solving for vertex points  $x_I$ , below (See sections 4.2.1 and 4.2.5);  $V = \{\};$  $V \leftarrow V \bigcup x_I, \ \forall \ x_I = A_I^{-1} \tilde{b}_I, \ I \subset I_m, \ |I| = d \ // \text{ for combinations of rank}(A_I)$ = d: for  $x_I \in V$  do // Constructing upper cell point  $p_{x_I}$ , below (See section 4.2.2); Form  $I \subset I_m$  and  $A_I \in \mathbb{R}^{d \times d}$  for current  $x_I$ ; **Run Subroutine:**  $p_{x_I} \leftarrow \text{algorithm}_9(I, x_I, A_I);$ // Forming and unpriming  $\alpha'_I$ , below (See sections 4.2.1 and 4.2.4); Form  $I' \leftarrow \forall i \in I$  s.t  $\sigma_i(p_{x_I}) \neq 1$ ;  $\alpha_I' = 0;$ for i = 1 to  $2^d$  do  $| \alpha'_I = \alpha'_I + g(\nu \in C_i)S_I(\nu \in C_i); // \text{ Equation (4.5)}$ end **Run Subroutine:**  $\alpha \leftarrow \text{algorithm}_{-10}(I, I', \alpha'_I, \alpha) // \text{Updates}$ coefficients of  $\alpha$ ; end // Projecting arrangement down to d-1, below (See section 4.2.3); Find  $x_I^m \in V$ ; // Minimum vertex w.r.t direction  $e_d$ ;  $x_F[d] = x_I^m[d] //$  fixed coordinate value;  $\tilde{b} \leftarrow \tilde{b} - A[:, d] x_F[d]; // A[:, d] \in \mathbb{R}^{m \times 1};$  $A \leftarrow A[:, 1: d-1];$ end  $\alpha_{\emptyset} = \alpha_{\emptyset} + g(Ax_F - b);$  // Equation (4.5) for constant term  $I = \emptyset$ ; **Return:**  $\alpha$ ;

Algorithm 11: Efficient Computation of an  $\alpha$ -vector

The GPU implementation is written at a warp level to solve 32 coefficients of an  $\alpha$ -vector simultaneously. Each block uses shared memory to load  $A, \tilde{b}$  in from global memory, at which point 32 threads evaluate 32  $\alpha'_I$ . This means that each thread requires shared memory space for an  $A_I$ , as well as space required to thread-wise take the inverses of the  $A_I$ , find the upper cell, and compute the *g*-values. These requirements make shared memory precious. As  $A_I$  are small, the warp uses Gaussian Elimination at the thread level, solving 32 'inverses' simultaneously.

The key to the efficient GPU implementation of algorithm 11 is the use of (GPU device side) hash tables to cache g-values across warps and blocks that are constructing elements of the same  $\alpha_i^{k|k}$ . Evaluating the inner-most for loop (of  $2^d$  g-evaluations) is by far the most expensive step of algorithm 11, and requires evaluating the left and right-hand side numerators of (3.20) as algorithm 8 does to compute  $g(\nu \in C_i)$ . This computation is ameliorated tremendously by the hash table.

To emphasize this point, a vector  $\alpha$  requires  $s_g(m,d) * 2^d$  g-evaluations, however, there are only maximum  $s_c(m,d)$  g-values. For example, a 10 hyperplane arrangement in 3 dimensions would require  $s_g(10,3) * 2^3 = 1408$  g-evaluations. However, there are only 92 (max) g-values. Using the hash table strategy saves 1316 g-computations. Further savings could be had if all children belonging to the same parent  $\alpha_i^{k|k-1}$  cached their left and right-hand side numerators of (3.20) in the hash table. To be exact, the savings would be an exorbitant  $2 * (m + 1) * s_g(m, d) * 2^d - s_c(m, d)$  g-evaluations. This was not endeavored, however, due to the tremendous programming complexity required to track and load all children of a parent (after the coalignment step) in the same kernel launch. Once the  $\alpha'_I$  coefficients have been evaluated (and unpriming is complete), the block moves onto another grouping of 32 coefficients, utilizing the same hash table for g-lookups, until the entire  $\alpha$ -vector has been computed.

### 6.2.9 Term Reduction

Only arrangements with the same number of hyperplanes are compatible for term reduction. Let  $A^m$  be the set of all arrangements of terms with m hyperplanes, with N(m)being the number of arrangements in this set. Let  $b^m, \bar{y}^m, \alpha^m$  be the accompanying set of parameters to these arrangements. For convenience, the superscript k|k is dropped. Then, the term reduction routine can be given as // We complete the following algorithm over all  $m \in M$ ; **Input:**  $A^m, b^m, \bar{y}^m, \alpha^m;$ Set:  $\epsilon \in \mathbf{R}_+$ , to a small positive number; Set:  $\mathscr{F} \in \mathbf{B}^{N(m)}$  to all True; for i = 1 to N(m) - 1 do if  $\mathscr{F}_i$  then // Step 1: Store indices j which meet criterion below in index set  $J_i$ ; for j = i + 1 to N(m) do Assert  $|b_i^m - b_j^m| < \epsilon$ , else remove j from  $J_i$  // Only check j's where  $\mathscr{F}_{i} = \text{True};$ Assert  $|\bar{y}_i^m - \bar{y}_i^m| < \epsilon$ , else remove j from  $J_i //$  Only check j's where  $\mathscr{F}_i = \text{True};$ end // Step 2: Check arrangements in  $J_i$ ; Declare  $\sigma \in \{\pm 1\}^{len(J_i) \times m}$ ; for q = 1 to  $len(J_i)$  do  $\mathbf{j} = J_i[q];$ for k = 1 to m do if  $|A_{ik}^m - A_{jk}^m| < \epsilon$  then  $\sigma[q,k] = 1;$ else if  $|A_{ik}^m + A_{jk}^m| < \epsilon$  then  $\sigma[q,k] = -1;$ else // Check Failed (CF), terms i, j don't reduce; remove j from  $J_i$  (after Step 2 for loop); remove row q from  $\sigma$  (after Step 2 for loop); end end end // Step 3: Term combinations; //  $\sigma$  and  $J_i$  are updated for CF above; for q = 1 to  $len(J_i)$  do  $j = J_i[q];$  $\hat{\alpha} = S(\sigma[q,:]) / S$  expansion of opposing normals of  $A_i$  and  $A_j$ ;  $\alpha_i = \alpha_i + \alpha_j \circ \hat{\alpha};$ end  $\mathscr{F}_{J_i}$  = False // Step 4: Set Flags of terms which reduced with *i* to False; else // Term *i* has been reduced, no action needed; end end **Return:**  $A^m, b^m, y^m, q^m, \alpha^m$  for True indexes of  $\mathscr{F}$ ; Algorithm 12: Term Reduction Algorithm

The runtime of algorithm 12, to leading order, is  $\mathcal{O}(N(m)^2)$ , which is quadratic in the number of terms of a particular arrangement size. Here, blocks of  $8 \times 32$  threads are declared. Each block is allocated to check chunks of 128 terms. The first block will check terms 0-127 against 0-127. The second block will check terms 0-127 against 128-255, and so on. Similar to algorithm 12, the blocks of the GPU kernel use the  $b^m$ ,  $\bar{y}^m$  to weed out which  $A^m$  elements in the chunk to check for reduction, since all three conditions must be met to reduce the terms. For all candidates, a row (warp) of threads checks one  $A_i$ against one candidate  $A_j$ , and updates the  $\alpha_i$  if the arrangements are  $\epsilon$ -close. This allows for 8 comparisons of arrangements to run simultaneously within each thread block.

# Chapter 7

# Extended Multivariate Cauchy Estimator for Nonlinear Dynamical Systems

Since the MCE is formulated for linear systems with additive Cauchy noise, the extension to nonlinear systems follows the methodology of the extended Kalman filter. Consider the discrete-time nonlinear dynamic system as:

$$x_{k+1} = f(x_k, u_k) + \Gamma_k w_k \tag{7.1}$$

$$z_k = h(x_k) + v_k, \tag{7.2}$$

where the state  $x_k \in \mathbb{R}^n$ , the elements of the initial state  $x_1$  are independent and Cauchy distributed,  $u_k \in \mathbb{R}^q$  is a deterministic control, f, h are nonlinear functions,  $\Gamma_k \in \mathbb{R}^{n \times r}$ is a weighting matrix on the independent vector of process noise  $w_k \in \mathbb{R}^r$  and  $z_k \in \mathbb{R}^p$ is the measurement model with an independent Cauchy noise vector  $v_k \in \mathbb{R}^p$ . Note that (7.1) is the nonlinear version of (2.1) without the added control input.

To derive the extended form of the Cauchy estimator, we assume that an estimate  $\hat{x}_k \approx \mathbb{E}[x_k|y_k]$  at step k given the measurement history  $y_k$  as defined in chapter 2 has been obtained and the estimate at k + 1 is to be determined. Projecting the posterior

state estimate  $\hat{x}_k$  to k+1, using the system dynamics of (7.1), is the a priori state

$$\mathbb{E}[x_{k+1}|y_k] \approx \bar{x}_{k+1} = f(\hat{x}_k, u_k). \tag{7.3}$$

The state variation can be formed by comparing the system dynamics (7.1) with the propagation of the conditional mean (7.3), to yield

$$\delta x_{k+1|k} = x_{k+1} - \bar{x}_{k+1} = f(x_k, u_k) - f(\hat{x}_k, u_k) + \Gamma_k w_k$$

$$\approx \nabla f_x(x_k, u_k) \Big|_{\hat{x}_k, \bar{u}_k} \delta x_{k|k} + \nabla f_u(x_k, u_k) \Big|_{\hat{x}_k, \bar{u}_k} \delta u_{k|k} + \Gamma_k w_k$$

$$= \Phi_k \delta x_{x|k} + \Gamma_k w_k, \qquad (7.4)$$

with  $\delta x_{k|k} = x_k - \hat{x}_k$  and  $\delta u_{k|k} = u_k - \bar{u}_k$  for a known control input  $\bar{u}_k$ . Equation (7.4) is equivalent to a first-order Taylor series expansion of (7.1) about the estimate  $\hat{x}_k$  and a control  $\bar{u}_k$ . Since  $u_k$  is assumed to be known, we set  $u_k = \bar{u}_k$  and thus  $\nabla f_u(x_k, u_k) \Big|_{\hat{x}_k, u_k} \delta u_{k|k} = 0$ . Consequently, the control input  $u_k$  will affect only the deterministic term of the estimator,  $\bar{x}_{k+1} = f(\hat{x}_k, u_k)$ .

At k + 1, the perturbed measurement is constructed as

$$\delta z_{k+1} = z_{k+1} - \bar{z}_{k+1} = h(x_{k+1}) - h(\bar{x}_{k+1}) + v_{k+1}$$

$$\approx \nabla h_x(x_{k+1}) \Big|_{x_{k+1} = \bar{x}_{k+1}} \delta x_{k+1|k} + v_{k+1}$$

$$= H_{k+1} \delta x_{k+1|k} + v_{k+1}.$$
(7.5)

Using (7.4) as the dynamics and processing  $\delta z_{k+1}$  of (7.5) as the measurement, the MCE is to estimate  $\mathbb{E}[\delta x_{k+1|k}|y_{k+1}] = \mathbb{E}[x_{k+1} - \bar{x}_{k+1}|y_{k+1}]$ . The posterior state estimate is obtained by adding  $\bar{x}_{k+1}$  and  $\mathbb{E}[x_{k+1} - \bar{x}_{k+1}|y_{k+1}]$  as

$$\hat{x}_{k+1} \approx \mathbb{E}[x_{k+1}|y_{k+1}] = \bar{x}_{k+1} + \mathbb{E}[x_{k+1} - \bar{x}_{k+1}|y_{k+1}] 
= \mathbb{E}[x_{k+1}|y_k] + \mathbb{E}[x_{k+1}|y_{k+1}] - \mathbb{E}[\bar{x}_{k+1}|y_{k+1}] 
= \mathbb{E}[x_{k+1}|y_k] + \mathbb{E}[x_{k+1}|y_{k+1}] - \mathbb{E}[\mathbb{E}[x_{k+1}|y_k]|y_{k+1}] 
= \mathbb{E}[x_{k+1}|y_k] + \mathbb{E}[x_{k+1}|y_{k+1}] - \mathbb{E}[x_{k+1}|y_k] 
= \mathbb{E}[x_{k+1}|y_{k+1}].$$
(7.6)

The additional computation to extend the MCE to nonlinear systems is in constructing the Jacobian matrices  $\Phi_k \in \mathbb{R}^{n \times n}$  and  $H_{k+1} \in \mathbb{R}^{p \times n}$  at each time-step k.

The parameters of the characteristic function in the MCE must now be updated to account for adding  $\mathbb{E}[\delta x_{k+1|k}|y_{k+1}]$  to the a-priori estimate  $\bar{x}_{k+1}$  to form  $\hat{x}_{k+1}$  as given in (7.6). It can be easily shown that the characteristic functions of random variables  $x \in \mathbb{R}^n$ and y = x + c, where  $c \in \mathbb{R}^n$  is a known constant vector, are related as [38]

$$\Phi_Y(\nu) = e^{j\nu^T c} \Phi_X(\nu). \tag{7.7}$$

An inspection of (2.6d) and (7.7) reveals that to properly account for  $\mathbb{E}[\delta x_{k+1|k}|y_{k+1}]$ when constructing the characteristic function (of the prior) at k + 1, the term  $c = -\mathbb{E}[\delta x_{k+1|k}|y_{k+1}]$  will need to be added to the parameters  $b_i^{k+1|k+1} \in \mathbb{R}^n$  for all  $i \in [1, ..., N_t^{k+1|k+1}]$ . Algorithm 13 below shows how to deal with the case when more than one measurement is used, i.e.,  $z \in \mathbb{R}^p$  for p > 1. It is usually advantageous to relinearize the measurement model  $h(\cdot)$  around the updated state estimate  $\hat{x}_k$ . That is, after processing measurement  $z_k[j], j \in [1, ..., p-1]$  and forming the estimate  $\hat{x}_k^{(j)}$ using  $H_k^{(j)} = \nabla_x h(\hat{x}_k^{(j-1)})[j, :]$  (and where  $\hat{x}_k^{(0)} = \bar{x}_k$ ), re-linearize  $h(\cdot)$  about  $\hat{x}_k^{(j)}$  as  $H_k^{(j+1)} = \nabla_x h(\hat{x}_k^{(j)})[j+1, :]$  for processing the measurement  $z_k[j+1]$ . If one is less certain that the updated state  $\hat{x}_k^{(j)}$  will lead to a better gradient calculation for  $H_k^{(j+1)}$ , the previous gradient can be used instead. Also, note that for multiple measurements, the time propagation step is only run for the first measurement (to take the system from k to k + 1). The results are summarized for a single time step below.

// EMCE for Estimation Step  $k \rightarrow k + 1$ ; **Input:** State estimate  $\hat{x}_k$ , control  $u_k$ , new measurement (vector)  $z_{k+1} \in \mathbb{R}^p$ ; Note: Measurement history is  $y_{k+1} = \{z_1, \dots z_{k+1}\};$ Set:  $\Phi_k = \nabla f_x(x_k, u_k) \Big|_{\hat{x}_k, \bar{u}_k}, \ \Gamma_k = \nabla f_u(x_k, u_k) \Big|_{\hat{x}_k, \bar{u}_k};$ **Propagate:**  $\bar{x}_{k+1} = f(\hat{x}_k, u_k);$ Set:  $k \leftarrow k + 1, \ \hat{x}_k^{(0)} = \bar{x}_k \ ;$ for j = 1 to p do  $\begin{aligned} \mathbf{Set:} \quad &\delta z_k^{(j)} = z_k[j] - h(\hat{x}_k^{(j-1)}), \\ &H_k^{(j)} = H_k[j,:] \in \mathbb{R}^{1 \times n}, \quad H_k = \nabla h_x(x_k) \Big|_{\hat{x}_k^{(j-1)}} \in \mathbb{R}^{p \times n}; \end{aligned}$ if j == 1 then //Full Step: Time Propagation and Measurement Update; **Run:**  $\left(\mathbb{E}[\delta x_{k|k-1}|y_k], P_k^{(j)}\right) = \text{MCE}(\Phi_{k-1}, \Gamma_{k-1}, H_k^{(j)}, \delta z_k^{(j)});$ else //Measurement Update Step Only; **Run:**  $\left(\mathbb{E}[\delta x_{k|k-1}|y_k], P_k^{(j)}\right) = \text{MCE}_{\text{MU}}(H_k^{(j)}, \delta z_k^{(j)});$ end // Update the Characteristic Function of the MCE  $(\Phi_Y(\nu));$ Update:  $\Phi_Y(\nu) = e^{j\nu^T c} \Phi_X(\nu), \quad c = -\mathbb{E}[\delta x_{k|k-1}|y_k];$ // Refine the State Estimate; **Update:**  $\hat{x}_{k}^{(j)} \leftarrow \hat{x}_{k}^{(j-1)} + \mathbb{E}[\delta x_{k|k-1}|y_{k}];$ end

**Return:**  $\hat{x}_{k}^{(j)}, P_{k}^{(j)};$ 

Algorithm 13: Extended Cauchy Estimator for Nonlinear Systems and Multiple Measurements

# Chapter 8

# Experiments

This chapter discusses the performance and execution speeds of the MCE and EMCE for several experiments in Cauchy, Gaussian, and  $S-\alpha-S$  noise. A simple three-state linear simulation in Cauchy noise is first provided to illustrate the advantages of the proposed estimator within impulsive noise environments. Execution speeds are then benchmarked for the methods proposed in chapters 3 and 4 for both the serial and distributed implementations of the MCE. Next, a nonlinear three-state homing missile simulation is given, using simulated  $S-\alpha-S$  measurement radar noises suggested by [9]. Monte Carlo simulations demonstrate the robustness the proposed estimator has for the  $\alpha \in [1, 2]$  family of densities. Lastly, a five-state low earth orbit satellite simulation is given, showing the application of this estimator to challenging real-world scenarios.

Experiments were conducted on an Asus Strix laptop computer with an Intel i7-6700HQ CPU clocked at 2.60GHz and an NVIDIA 1060 Geforce GTX GPU. In addition, the National Science Foundation XSEDE Expanse GPU cluster was also used to test the application on the more powerful NVIDIA V100 SMX2 GPU.

# 8.1 A Linear Three-State Simulation in Cauchy Noise

## 8.1.1 Formulation

The performance of the MCE against the Kalman filter for the three-state linear dynamic system (previewed in section 3.5) in both a Cauchy and Gaussian noise simulation is given. The dynamic system is

$$\Phi = \begin{bmatrix} 1.40 & -0.60 & -1.00 \\ -0.20 & 1.00 & 0.50 \\ 0.60 & -0.60 & -0.20 \end{bmatrix}, \ \Gamma = \begin{bmatrix} 0.10 \\ 0.30 \\ -0.20 \end{bmatrix}, \ H = \begin{bmatrix} 1.00 \\ 0.50 \\ 0.20 \end{bmatrix}^T, \ \beta = 0.1, \ \gamma = 0.2 \quad (8.1)$$

where  $\beta$  is the scaling parameter of the Cauchy pdf of the process noise, and  $\gamma$  is the scaling parameter of the Cauchy pdf of the measurement noise. For the Kalman filter, the corresponding noise terms are simply  $1.3898\beta$  and  $1.3898\gamma$  using the least squares fitting result of section 1.4. We compare the performance of the two estimators in both a Cauchy and Gaussian noise simulation.

The dynamic system of (8.1) was first used to showcase the MCE algorithm in [16], which used the backward recursive characteristic function presented in chapter 2. Until now, the sliding window approximation had not been discovered for three-state problems. Therefore, applications involving three states or more could not be pursued. Moreover, propagating the backward recursive characteristic function as in chapter 2 was simply too expensive, as will be shown. The sliding window approximation was, however, applicable to the special case of two state problems [15, 31], but was exceptionally slow due to the method of chapter 2. By using either the method of chapter 3 or chapter 4 to compress the characteristic function, the compressed MCE can now use the sliding window approximation like in [15, 31], but now for multivariate systems and with a remarkably reduced number of terms encompassing the characteristic function at each step k, when compared to [16]. Furthermore, parallel programming is seen to be highly advantageous for the compressed MCE. The estimation algorithm can now run at execution speeds which, for many applications, would be considered real-time appropriate.

# 8.1.2 Numerical Results



Figure 8.1: Comparison of the MCE to a Kalman filter in a Gaussian and Cauchy noise simulation. Cauchy errors are primarily bounded by their one-sigma confidence bound in Cauchy noise, whereas both estimators are bounded by their one-sigma values in Gaussian noise.

Figure 8.1 (left column) illustrates the interesting properties of the MCE in a Cauchy noise simulation. As given in (2.9), the variance of a Cauchy estimator is explicitly a function of the measurement, and therefore the conditional variance dynamically adjusts to the measurement history. This is not the case in the Kalman filter, where the filter's posterior covariance is a-priori known. At k = 3, a pulse occurs in the process noise. We see that the Cauchy Estimator tracks this jump in all three states with ease, while the Kalman filter mostly ignores this pulse in states one and three, producing large state errors. At k = 4, a pulse occurs in the measurement noise. We see the MCE ignores this pulse while sharp jumps are seen in the Kalman filter.

This behavior observed in Fig. 8.1 is the result of the MCE's ability to hypothesize multi-modal beliefs of the conditional state estimate, as was depicted in Fig. 1.1. Depending on the magnitudes of the process and measurement scaling parameters  $\beta$ ,  $\gamma$ , the estimator can quickly hedge between whether outliers are more likely to be the result of process or measurement noises. Above, the process noise impulse was smaller than the measurement noise impulse, to an appropriate level. The MCE correctly tracked the process noise outlier and ignored the measurement noise outlier. If the realization of the process outlier happened to be much larger, the MCE would not track the jump in the process (as  $\gamma > \beta$ ), momentarily concluding that the sensor measurement outlier must be due to the measurement noise. However, it would also hedge this bet by allotting a non-negligible amount of probability to the chance the outlier was in fact due to the process. If the next sensor measurement is consistent with that of the prior estimation step, the estimator will quickly track the jump and revert its hedge. This behavior is consistent with what is seen in particle filtering methods, however, these methods require large particle sets to produce this exact behavior [7].

Figure 8.1 (right column) shows the performance of both estimators in Gaussian noise simulation. We see clearly, in Gaussian noise, the Kalman filter is the superior estimation scheme. It is interesting to note how the one-sigma values of the MCE upper-bound those of the Kalman filter and that in Gaussian noise, both of the estimator's one-sigma values bound the estimation error at each step.

Reducing terms is seen to have a tremendous computational advantage over not reducing terms at the end of each estimation step k. Table 8.1 illustrates the savings. At step k = 8, the new estimation scheme encompasses only 1% of the former number of terms required. This is a core advantage of the compressed MCE algorithm over the method of chapter 2.

Table 8.1: Number of terms eliminated by the term reduction algorithm at each estimation step. Depicted are the results of reducing terms for the three-state example.

Time Step (k)	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8
No Term Reduction	4	20	120	792	$5,\!440$	37,936	216,066	$1.69 \mathrm{M}$
With Term Reduction	4	14	48	161	542	1,762	5,709	18594

#### Execution Speeds

The three-state problem of (8.1) is now used to demonstrate the computational improvements of the MCE since its inception in [16]. Execution times for arbitrary three-state dynamic systems follow very closely to the execution speeds presented here. The execution times presented are significant because they are identical to the execution times of the sliding window approximation of chapter 5 for a respective window length. Put another way, the execution speed of the sliding window method for W sliding windows is solely dependent upon how quickly the W-th step of the estimator can be run (provided sufficient computational resource is available for all windows). The 'Hz-Rate' columns in the following tables are provided for this reason.

To showcase the (remarkable) improvement in execution time the MCE has undergone, Table 8.2 first presents the original implementation of the MCE used in [16], which implements the backward recursive characteristic function of chapter 2. Table 8.2 also shows the Python implementation of the MCE, which originally tested the method of chapter 3 and allowed for similar terms to reduce together. The impact that reducing terms had was clear, lowering the time required to run 8 estimation steps by 108 times.

Table 8.2: The original 'backward-recursive' Matlab MCE and the prototype Python implementation of the MCE using the method of chapter 3.

Language	Estimation Step	Step Time (sec)	Hz-Rate
Matlab	6	23.20	0.0431
Matlab	7	317.46	0.00315
Matlab	8	8762.37	0.000114
Python	6	5.01	0.1996
Python	7	18.92	0.0528
Python	8	80.94	0.0123

Next, a C/C++ implementation of the MCE was pursued to test the performance of both chapters 3 and 4 methods without (Python) interpreter overhead. Table 8.3 depicts the results of the serial C implementation. The C-code was designed to be a singlethreaded application, executing on a single CPU. The LAPACK linear algebra library was used to solve the least squares problem of (3.6), while the GNU linear programming kit (GLPK) was used to solve the small phase-I feasible linear programs required by Inc-Enu. The C code was seen to reduce the interpreter overhead at step 8 by 20 times. The method of chapter 4 halved the overall run-time when compared to that of chapter 3.

Listed in Table 8.4 is the amount of time required by using either the chapter 3 or chapter 4 method to parameterize the  $\alpha$ -vectors, for the terms at step 8. The advantage

Method	Estimation Step	Step Time (sec)	Hz-Rate
Chapter 3	6	0.201	4.975
Chapter 3	7	0.847	1.181
Chapter 3	8	4.199	0.238
Chapter 4	6	0.117	8.547
Chapter 4	7	0.500	2.00
Chapter 4	8	2.023	0.493

Table 8.3: The C/C++ implementation of the MCE using the methods of chapters 3 and 4. Times are reported for a single thread running on a single processor.

of this method is clear as the number of hyperplanes in the arrangement becomes larger. Results are similar for the CUDA-C implementation. See section 6.2.8 for comments on how this speed-up could be significantly improved.

Table 8.4: Comparing the C/C++ execution speeds to parameterize  $\alpha$  using the methods of chapters 3 and 4. Note ms is shorthand for milliseconds.

Number of Hyperplanes	Terms	Chap. 3 Time (ms)	Chap. 4 Time (ms)	Speed-Up Factor
4	6223	318	248	1.28
5	2685	312	189	1.65
6	1453	302	174	1.74
7	994	337	192	1.76
8	780	408	210	1.94
9	2484	2330	965	2.41

A CUCA-C implementation of the MCE was then devised to test the true computational performance of both chapters 3 and 4 methods by using GPU parallel compute. The CUDA-C implementation conducts all subroutines enumerated in chapter 6 on the GPU, maximizing the computational throughput of the full MCE algorithm. Optimizing this performance required significant development time. Shown in Table 8.5 are the results of the GPU parallelization. For the method of chapter 3, the GPU implementation on the modest 1060 GeForce GTX is able to execute step 8 now in 179 milliseconds, a speed-up over the CPU version of 23.5 times. For the same GPU, the MCE is now able to execute step 8 now in 85 milliseconds using the technique of chapter 4, gaining a speed-up of 23.8 times over that of its CPU implementation. Compared to running 8 steps using the original MCE Matlab implementation, the GPU code is now ~50,000 times faster with the chapter 3 method and ~ 100,000 times faster using the chapter 4 method.

Method	Step	Step Time (ms) on GPU 1060	Hz-Rate GPU 1060	Step Time (ms) on GPU V100	Hz-Rate GPU V100	Number of terms
Chapter 3	6	10.13	98.71	3.16	315.6	1762
Chapter 3	7	38.90	25.70	7.50	132.83	5709
Chapter 3	8	179.8	5.56	24.16	41.378	18594
Chapter 4	6	5.88	169.98	2.36	423.01	1762
Chapter 4	7	18.09	55.27	3.96	252.20	5709
Chapter 4	8	85.25	11.73	10.2	98.52	18594
Chapter 4	9	-	-	42.6	23.45	80288
Chapter 4	10	-	-	275.4	3.63	326497

Table 8.5: The CUDA-C implementation of the MCE using the methods of chapters 3 and 4

The results are even more impressive when utilizing a top-of-the-line GPU, such as the V100. For the method of chapter 3 executes step 8 now in 24 milliseconds, a speed-up over the CPU version of 173.8 times. For the same GPU, the MCE is now able to execute step 8 now in 10 milliseconds using the technique of chapter 4, gaining a speed-up of 200 times over that of its CPU implementation. Compared to running 8 steps using the original MCE Matlab implementation, the GPU code is now  $\sim$  365,000 times faster using the chapter 3 method and  $\sim$  860,000 times faster using the chapter 4 method. Execution times for steps 9 and 10 are also provided for the chapter 3 method, showing the V100 can still run sub-second for hundreds of thousands of terms. Note that execution speeds for window sizes 9 and 10 require close to six gigabytes of device memory to achieve real-time performance and therefore are not given for the 1060 GPU due to its memory limitation. It should also be mentioned that because the software has been optimized for larger window sizes (i.e., seven through ten), program launch parameters for smaller window sizes could truly be optimized further.

#### Execution Speed for LTI Systems

Discussed in section 3.7 was the special case when a system is linear time-invariant. The theory presented in [22] shows that the indices at which hyperplanes coalign (termcoalignment) and the indices at which terms reduce with one another (term-reduction) are a-priori known, regardless of the initial conditions provided. The majority of the compute within the term-coalignment and term-reduction sub-routines can be removed for this special case. Specifically in term-coalignment (for each term), hyperplanes at the known indices of coalignment are discarded. In term reduction, the  $\alpha_i^{k|k}$  vectors of terms at the known indices of reduction must simply be added together. The bulk of the computation savings is within the term-reduction sub-routine, where the quadratic run-time is now effectively linear in the number of (a-priori known)  $\alpha_i^{k|k}$  vector additions.

Table 8.6 shows the execution speeds of the CUDA-C implementation for the LTI example given, using this a-priori information. We see a 40% increase in execution speed for the method of chapter 4 at step 8. This is because the method of chapter 4 for computing the  $\alpha$ -vectors is much quicker than that of chapter 3, and therefore removing the majority of compute from term-reduction has greatly improved the overall run-time. The results of step 7 indicate large savings are not observed until the characteristic function of the MCE is made up of greater than 10000 terms (i.e., step  $\geq$  8). We see that for steps 9 and 10, the V100 GPU yields approximately a 2 or 3 time speed increase over those in Table 8.5, respectively. Note that the number of terms is the same as in Table 8.5.

Table 8.6: The CUDA-C MCE using the methods of chapters 3 and 4 with a-priori term-coalignment and term-reduction

Method	Estimation Step	Step Time (ms) on GPU 1060	Hz-Rate on GPU 1060	Step Time (ms) on GPU V100	Hz-Rate on GPU V100
Chapter 3	7	38.90	27.78	7.4	144.38
Chapter 3	8	142.8	7.00	19.3	51.73
Chapter 4	7	15.75	63.75	3.448	290.03
Chapter 4	8	50.52	19.41	6.04	165.43
Chapter 4	9	-	-	20.87	47.906
Chapter 4	10	-	-	107.4	9.31

#### **Discarding Terms**

As discussed in section 3.7.1, a numerical study of the terms of the characteristic function reveals that for many terms, the norm of its  $\alpha_i^{k|k}$  parameter vector approaches zero. If a small value of  $\epsilon$  is chosen, terms of the MCE with  $||\alpha_i^{k|k}|| \leq \epsilon$  could be discarded, at the loss of slight numerical precision in the mean/variance calculation. Here, the 1-norm is used. Shown in Table 8.7 are the number of terms removed by this approximation at step 8. The maximum deviation from the conditional mean/variance when compared with the estimate without the approximation is given next. The maximum complex error associated with (2.8) and  $(2.9)^1$  are then given, followed by the computation time taken.

Epsilon	Terms removed (of 18594)	Mean Deviation	Max Mean Imaginary Part	Variance Deviation	Variance Imaginary Part	Time (ms) GPU 1060
0	0	-	8e-15	-	6e-12	85.25
1e-19	1317	1e-16	2e-13	2e-12	1e-11	80.63
1e-18	1958	2e-13	3e-12	7e-11	3e-11	77.76
1e-17	2562	3e-12	1e-11	2e-10	5e-10	75.31
1e-16	3551	7e-10	4e-10	3e-9	8e-9	74.48
1e-15	4370	7e-10	4e-10	2e-9	7e-9	72.71
1e-14	5609	1e-9	1e-9	2e-8	6e-8	68.87

Table 8.7: Discarding terms of the MCE at step 8, given  $||\alpha_i^{k|k}|| \leq \epsilon$ .

The alpha approximation presented in Table 8.7 could be used in conjunction the a-priori knowledge of LTI systems, however this was not attempted. Doing so, would likely increase the execution speeds in Table 8.6 slightly, possibly up to 10-15%. It is also interesting to note that if the initial condition  $p^{1|0}$  for the MCE was made equal to a zero vector, indicating that the initial state is a-priori known, a massive 17192/18594 terms are removed by step 8, using just  $\epsilon = 1e-19$ . Error statistics for this case are very similar to those of row two in Table 8.7. Unfortunately, this is not pursued since the initialization method of chapter 5 will not create  $p^{1|0}$  equal to zero when constructing the initialization parameters  $\{A^{1|0}, b^{1|0}, p^{1|0}\}$ . It should be noted however, that if the initialization method proposed in chapter 5 was removed, and a window size of ~ 8 was chosen, the performance of the estimator does not degrade very much, and the execution speed of 8-steps is approximately twice the Hz-rate reported in Table 8.5 for step 6 (i.e, 850Hz on the V100 and 330Hz on the 1060). These experiments indicate that there is likely further work that can be explored regarding the  $||\alpha_i^{k|k}|| \leq \epsilon$  approximation, its empirical performance, and its numerical stability.

<sup>&</sup>lt;sup>1</sup>The mean and variance are calculated as complex numbers, but should have only real value and a zero imaginary part. Due to numerical imprecision in the computation of (2.8) and (2.9), this is not the case, but should be near zero.

#### Windowing Example

Compared in Fig. 8.2 is the state error and one-sigma confidence bounds of the Kalman filter and the MCE. The simulation uses the Cauchy noise simulation realization depicted in Fig. 8.3 and the dynamic system of (8.1). The MCE uses the sliding window approximation of chapter 5. The experiment was repeated for each window size 4 through 8, using this same process and measurement noise realization of Fig. 8.3 to test the impact of window size on estimation performance. Note that Figure 8.4 (presented later) gives a zoomed-in view of the state errors and one-sigma confidence bounds in Figure 8.2.



Figure 8.2: State error plot of the MCE for window sizes of four through eight, compared to the Kalman filter, in a Cauchy noise simulation. Solid lines depict the state estimation error of the MCE, while dashed lines depict the one-sigma confidence bound of the estimators.

The MCE's ability to quickly track large jumps in the state is clear in Fig. 8.2. The process noise realization (blue) that is depicted in Fig. 8.3 shows where these jumps occur. Although  $\beta < \gamma$ , it is seen in Fig. 8.3 that the process noise realizations at steps 27 and 55 are very large. We see in Fig. 8.2 that the MCE ignores these process jumps at 27 and 55, concluding (momentarily) that these outliers are most likely due to a large measurement noise entering the system (as  $\beta < \gamma$ ). At the next estimation step, however, the sensor measurement is consistent with the prior measurement, and the MCE makes



Figure 8.3: Simulated measurement realization, along with the process and measurement Cauchy noise realizations.

a large update to its conditional state estimate and immediately tracks the jump. This result indicates that the MCE has now jumped to its second belief: the outlier was due to large process noise. This behaviour is clear by comparing the MCE to the Kalman filter. We see the Kalman filter takes *many* additional steps to correct for the large jumps at 27 and 55. This is the power of an estimator which does not simply use a linear gain that operates on measurement residuals, as the Kalman filter does.

Figure 8.4 zooms in on the state-error plot of Fig. 8.2. It is easier to see here that the MCE tracks the jumps quickly at steps 28 and 56. It is also interesting to note that at each time step where a large spike occurs in either the process or the measurement noise realization, the estimator adjusts its covariance (and thus, the one-sigma confidence bounds) dramatically. This variance information can be useful for two reasons. First, it shows exactly where the estimator is less certain of its belief, which could be useful for event-detection applications. Second, this information can be especially useful for applications which make use of a pdf or second moment statistics when forming a control law [39, 40]. Without even viewing Fig. 8.3, it is easy to tell exactly where either a volatile process or measurement noise realization has entered the system; by viewing the one-sigma confidence bound of the MCE. This is in contrast to the Kalman filter in Figs. 8.2 and 8.4, whose covariance estimate is not a function of the measurement.



Figure 8.4: Close-up of the state-error plot depicted in Fig. 8.2. All windows are seen to perform very similarly to one another.

Lastly, Fig. 8.4 shows how similar the conditional mean and variance estimates are using the sliding window approximation for window sizes 4 to 8. The state errors are seen to be practically indistinguishable for all windows. Moreover, the one-sigma confidence bounds appear to be almost identical. This implies that estimation accuracy and speed can be traded-off, to a degree, by using the sliding window approximation. With only a very slight loss in estimation accuracy, the window size of four is seen to be appropriate for this system. Running the sliding window approximation with multiple window sizes is a good way to trade-off estimation accuracy and execution speed, for a given application. For example, a sliding window approximation with six windows can achieve over 400 Hz on the V100 (see Table 8.5) and for four windows, can achieve over 1000 Hz (not listed, see [18]).

#### Two, Three, Four and Five State Problems

Table 8.8 gives a summary of the execution rates for dynamic systems of 2, 3, 4 and 5 dimensions, for the listed number of process and measurement noises. These times are

given for the 1060 GPU. For the V100, the rates would be in the range of 8-10 times those presented here. The execution rates is found by taking the reciprocal of the execution time, i.e, the time required to run the *n*-th step. This rates indicates the speed at which the slide window approximation of *n*-windows could continuously run a time-varying or nonlinear system of dimension 2, 3, 4, 5, respectively. Listed additionally is the execution rate if a-priori term-coalignment and term-reduction is used, i.e, if the system is indeed LTI. Note that 2, 3, etc. measurement noises implies that 2, 3, etc. measurements would be processed at each estimation step. Note that the times are given for the method of chapter 4, and are roughly half if the method of chapter 3 is used instead.

Table 8.8: Execution rates for systems of various dimensions on the 1060 GPU. Rates are given assuming the system is LTI, or not LTI. Note CF stands for characteristic function.

Dimension	Process Noises	Measurement Noises	Step Number	Rate (Hz) for Non-LTI System	Rate (Hz) for LTI System
2	1	1	8	150.0	222.2
2	2	1	7	70.2	161.2
2	2	2	6	1.5	26.8
2	2	2	5	31.1	122.0
3	1	1	8	11.7	19.4
3	2	1	6	23.4	25.8
3	1	2	5	14.3	33.7
3	2	2	4	17.1	28.6
3	3	1	5	34.8	46.8
3	3	2	4	2.8	5.8
3	3	3	3	2.62	4.4
4	1	1	6	20.7	33.3
4	2	1	5	15.5	16.5
4	1	2	5	14.9	20.8
4	2	2	4	1.0	1.3
5	1	1	6	24.0	26.8
5	2	1	4	33.3	35.8
5	1	2	4	1.45	1.6
5	2	2	3	4.3	4.5

An obvious result from Table 8.8 is the effect of using multiple measurement noises and process noises at each time step. The net effect is that the window size will need to decrease for such systems, as the computational and memory load can become large. Systems with n process noises (columns of  $\Gamma$ ) add n hyperplanes to each term's hyperplane arrangement per step. This in effect adds more computational effort to construct a term's  $\alpha$ -vector, as the dimension of  $\alpha$  will become larger as given by (3.2). Systems with n measurements will add many terms per estimation step, despite term reduction reducing many of them together. This is due to the factorial growth rate of the estimator when it generates child terms. The issue with child generation is that when there are many terms, the quadratic run-time of term reduction becomes large. This is clearly seen in the 2 dimension, 2 process noise, 2 measurement noise example. Many terms are created after processing 12 measurements (2 measurements/step × 6 steps), and the term reduction algorithm consume almost all of the time required. We see in the last column of this example that when the system is LTI, much of this work is removed, as we know a-priori which terms reduce, and therefore, term-reduction is effectively linear in the number of known reductions. In some of these larger systems, the window size of 4 or 3 is recommended to stay real-time. From the results of Fig. 8.4, we see that smaller window sizes (those about 4) do not really impact the performance of the estimator very much. This insight is additionally observed in the following experiment.

# 8.2 A Three-State Nonlinear Homing Missile Simulation in Heavy-Tailed Noise

# 8.2.1 Formulation

A nonlinear target-pursuer homing missile problem is now considered, modelled as suggested in [41] and depicted in Fig. 8.5. The pursuer's estimation objective is to use an onboard radar sensor to estimate the relative lateral position and velocity between itself and its target. The lateral acceleration of the target is estimated as well. The onboard radar provides a noisy line-of-sight bearing measurement between itself and the relative lateral and longitudinal distance to its target. These measurements are provided to the pursuer as it longitudinally closes in on the target and until the time-to-intercept reaches zero. The noise distribution associated with the radar sensor is modelled as a  $S-\alpha-S$  pdf with stability parameter  $\alpha = 1.7$  as suggested in [9], and with a time-varying fading, scintillation and glint noise model that is inversely proportional to the time-to-intercept [41]. The noises are generated using the method proposed in [19]. The target is modelled with non-Gaussian forcing evasive maneuvers. This is done by simulating the acceleration profile of the target as a telegraph wave with Poisson-distributed switching times. The telegraph (forcing) wave, although non-Gaussian, admits finite first and second moments. The telegraph forcing is modelled to have two modes: a long primary mode of  $\pm 3G$  amplitude and a shortly sustained secondary mode of  $\pm 9G$  amplitude, where G stands for the acceleration due to gravity.



Figure 8.5: Schematic of the three-state target-pursuer homing missile problem.

The lateral interception dynamics are modelled in continuous time as

$$\left\{ \begin{array}{c} \dot{y} \\ \dot{v} \\ \dot{a}_{T} \end{array} \right\} = \left[ \begin{array}{ccc} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & -\frac{1}{\tau} \end{array} \right] \left\{ \begin{array}{c} y \\ v \\ a_{T} \end{array} \right\} + \left[ \begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right] a_{p} + \left[ \begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right] w_{a_{T}}, \tag{8.2}$$

$$\frac{\dot{x}(t)}{\dot{x}(t)} = \begin{array}{c} F \\ \dot{x}(t) \end{array} = \begin{array}{c} 0 \\ \dot{x}(t) \end{array} \right] \left\{ \begin{array}{c} y \\ v \\ a_{T} \end{array} \right\} + \left[ \begin{array}{c} 0 \\ 1 \\ B \end{array} \right] a_{p} + \left[ \begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right] w_{a_{T}}, \tag{8.2}$$

where  $y, v \in \mathbb{R}$  are the relative lateral position and velocity between the pursuer and its target, respectively,  $a_T \in \mathbb{R}$  is the lateral acceleration of the target with a modeled first order lag coefficient  $\tau > 0$ ,  $a_p \in \mathbb{R}$  is the lateral acceleration of the pursuer, and  $w_{a_T} \in \mathbb{R}$ is additive white noise. The initial conditions on the lateral relative position and velocity are

$$\mathbb{E}[y(t_0)] = 0, \quad \mathbb{E}[v(t_0)] = 0,$$
  

$$\mathbb{E}[y^2(t_0)] = 0, \quad \mathbb{E}[v^2(t_0)] = P_v(0), \quad \mathbb{E}[y(t_0)v(t_0)] = 0.$$
(8.3)

The initial lateral velocity  $v(t_0)$  is random and is the result of launch error, whereas the initial lateral position  $y(t_0)$  is assumed to be known and zero. The controller for the pursuer  $a_p$  is set to be a guidance law that is a function of the true underlying simulation state instead of a function of the state estimates. This is done to limit the focus on state estimation performance and to avoid statements regarding state-estimate-based feedback control laws. This task is left to future work.

To be more realistic, the pursuer wishes to equate the statistics for its first-order target acceleration model in (8.2) to that of a telegraph wave with Poisson distributed switching times. In appendix A.2, it is shown that if the statistics of  $a_T$  in (8.2) are assumed Gauss-Markov with mean and auto-correlation given as

$$\mathbb{E}[a_T(t)] = 0, R_{a_T a_T}(t, s) = \mathbb{E}[a_T^2] \Phi(t-s) = \mathbb{E}[a_T^2] e^{-\frac{|t-s|}{\tau}},$$
(8.4)

where  $\Phi(t-s)$  is the transition matrix, then the Gauss-Markov model can be set equal to the second-order statistics of the telegraph wave, which are

$$\mathbb{E}[h_T(t)] = 0, \quad R_{h_T h_T}(t,s) = h_T^2 e^{-2\lambda |t-s|}.$$
(8.5)

This is done by setting  $\mathbb{E}[a_T^2] = h_T^2$  and  $\tau = \frac{1}{2\lambda}$  above, where  $\lambda$  is the Poisson rate parameter and  $h_T$  is the height of the telegraph. For simplicity, it is assumed the pursuer has a-priori knowledge of  $h_T$  and  $\lambda$  to equate the statistics of (8.4) and (8.5)

Let  $\Delta = t_{k+1} - t_k$  be the discrete-time increment with k the time index. The discrete-

time system of (8.2) is given by

$$\vec{x}_{k+1} = \Phi_k \vec{x}_k + B_k a_{p_k} + \Gamma_k w_k, \tag{8.6a}$$

$$z_k = \theta_k + n_k, \quad \theta_k = \arctan\left(\frac{y_k}{V_c(t_f - t_k)}\right),$$
(8.6b)

where

$$\Phi_{k} = \begin{bmatrix} 1 & \Delta & \tau^{2}(1 - e^{-\frac{\Delta}{\tau}}) - \tau \Delta \\ 0 & 1 & \tau(e^{-\frac{\Delta}{\tau}} - 1) \\ 0 & 0 & e^{-\frac{\Delta}{\tau}} \end{bmatrix}, \quad (8.6c)$$

$$\Gamma_{k} = \begin{bmatrix} \tau^{2}\Delta + \tau^{3}(e^{\frac{-\Delta}{\tau}} - 1) - \frac{1}{2}\tau\Delta^{2} \\ -\tau^{2}(e^{-\frac{\Delta}{\tau}} - 1) - \tau\Delta \\ -\tau(e^{-\frac{\Delta}{\tau}} - 1) \end{bmatrix}, \quad (8.6d)$$

$$B_{k} = \begin{bmatrix} \frac{1}{2}\Delta^{2} \\ \Delta \\ 0 \end{bmatrix}. \quad (8.6e)$$

The process noise  $w_k$ , approximating the telegraph wave maneuver of the target using a Gaussian model as derived in appendix A.2, is characterized by

$$w_k \sim \mathcal{N}\left(0, \frac{2}{\tau} \frac{h_T^2}{\Delta} \delta_{kl}\right),$$
(8.7)

where  $\delta_{kl}$  is the Kronecker-delta function. The radar measurement  $z_k$  of (8.6b) is the line-of-sight bearing  $\theta_k$  between the pursuer and its target, where  $V_c$  is the longitudinal closing speed of the pursuer onto the target and is assumed for simplicity to be constant, and  $t_f - t_k$  is the time-to-interception. The measurement noise  $n_k$ , as suggested by [41], is an additive, time-varying Gaussian with power spectral density components  $R_1, R_2$  that model fading, scintillation, and glint for the radar sensor. Its statistics are approximated by

$$E[n_k n_l] = V_k \delta_{kl} = \left[\frac{R_1}{\Delta} + \frac{R_2}{(t_f - t_k)^2 \Delta}\right] \delta_{kl}.$$
(8.8)

A linear guidance law (for small angles  $\theta_k$ ) is adopted for the pursuer from [41] as

$$a_{p_k} = -K_e V_c \dot{\theta}_k = -K_e \left[ \frac{(t_f - t_k)v_k + y_k}{(t_f - t_k)^2} \right],$$
(8.9)

where  $K_e$  is a proportional navigation constant set to 5. The simulation uses the simulated states to construct the control  $a_{p_k}$ . This allows the same state realization to be used for feedback by both estimators.

The homing missile simulation statistics stated in (8.7) and (8.8) are formulated as Gaussian (i.e.  $\alpha = 2.0$ ). However, we would like to simulate the radar measurement statistics of (8.8) as non-Gaussian with  $\alpha = 1.7$  [9]. To do so, we fit the underlying measurement statistics for  $\alpha = 1.7$  to the presented Gaussian form in (8.8). The statistics for the EKF follow those presented in (8.7) and (8.8). The process and measurement statistics for the EMCE ( $\alpha = 1.0$ ), however, must be fit to the Gaussians of (8.7) and (8.8) as well. Therefore, a least square fit for a pdf of  $\alpha = 1.7$  to a Gaussian ( $\alpha = 2.0$ ) pdf as well as a Cauchy ( $\alpha = 1.0$ ) pdf to the same Gaussian is performed by numerically minimizing (1.6). With the target pdf set Gaussian ( $\alpha_T = 2.0$ ) and the proposal distribution set to  $\alpha_P = 1.7$  yields the scale ratio  $\frac{\sigma_P}{\sigma_T} \approx 0.7096$ . Repeating the minimization for the Cauchy distribution ( $\alpha_P = 1.0$ ) fit to the Gaussian, we find a scale ratio of  $\frac{\sigma_P}{\sigma_T} \approx 0.7195$ , which is also confirmed analytically in [13, 16] by minimizing (1.5) for Cauchy and Gaussian pdfs. Note that the scale parameter for the Gaussian characteristic function is by definition  $\frac{\sigma}{\sqrt{2}}$ . Also note that the minimization yields a ratio that relates the scale parameters of the two distributions, and thus we set  $\sigma_T = 1.0$  for the minimizations above.

### 8.2.2 Experimental Simulations and Results

The simulation was conducted using  $\Delta = 0.1$ sec,  $\mathbb{E}[h_T^2] = 100^2$ ft<sup>2</sup>,  $\lambda = 0.75$ Hz,  $P_v = 200^2 \frac{\text{ft}^2}{\text{sec}^2}$ ,  $V_c = 300 \frac{\text{ft}}{\text{sec}}$ ,  $t_f = 10$ sec,  $R_1 = 15 \times 10^{-6}$ rad<sup>2</sup>sec and  $R_2 = 1.67 \times 10^{-3}$ rad<sup>2</sup>sec<sup>3</sup>. The interesting properties of the EMCE for nonlinear dynamics within a heavy-tailed environment is first discussed. For this experiment, the 1060 GPU was used. Next, the exciting robustness properties of the EMCE are illustrated through several Monte Carlo

experiments. For the numerically intensive Monte Carlo simulations, the V100 GPU cluster provided by XSEDE was used.

#### **Sliding Window Simulation**

For this experiment, we show the relative performance between an EKF and an EMCE for a single realization of the missile interception problem when an impulse in the measurement sequence is observed close to the intercept time at  $t_f = 10$  sec. Here, the pursuers guidance law  $a_{p_k}$  is set to zero. Doing so allows the measurement function to become more nonlinear towards the time-to-intercept. This enables a clear visualization of the advantage the EMCE holds over the EKF. The performance for EMCE window banks of size four to eight is compared against that of the EKF, within environmental noise with  $\alpha = 1.7$ , as discussed in section 8.2.1.

Figure 8.6 presents the simulation input data for the experiment. The top subplot of Fig. 8.6, shows the simulated line-of-sight bearing angle measurement provided to the pursuer by its on-board radar sensor. The middle subplot shows the  $S-\alpha-S$  radar noise with  $\alpha = 1.7$ . The bottom subplot is the acceleration profile of the target, simulated as a telegraph wave.

The performance of the EMCE and EKF for this sample run are presented in Fig. 8.7. Depicted are the state-estimation errors for the relative lateral position (top) and relative lateral velocity (middle) between the target and its pursuer. The lateral estimation error for the telegraph acceleration profile of the target is depicted in the bottom subplot. The dashed green line depicts the EKF's predicted one standard deviation confidence bound of its estimate, whereas the other dashed lines are the one standard deviation confidence bounds for the EMCE window banks. The horizontal axis for both Figs. 8.6 and 8.7 is the time-step, where at  $t = 10 \sec$  the pursuer catches up longitudinally with its target.

Figure 8.7 shows that both estimators appear to estimate the relative position and velocity similarly for the first 80 time-steps, while both struggle to determine the telegraph acceleration. As the pursuer closes in longitudinally, the nonlinearity of the measurement model will become more pronounced because longitudinal deviations at close distances



Figure 8.6: Simulated measurement, measurement noise, and process noise used in the target-pursuer experiment.

affect the bearing angle measurement more severely. Figure 8.6 shows that at timestep 84 when the pursuer is longitudinally close to its target, an impulse in the radar measurement noise is observed. The EKF reacts strongly to this impulse, while all EMCE window banks are seen to ignore the impulse. Due to the nonlinearity in the measurement model, the EKF is not able to recover from the outlier and its estimate of relative lateral position continues to diverge severely until the end of the experiment. This is caused by the fact that the linearization of the measurement model in the EKF is taken about the state estimate, which continues to worsen.

From time step 84 and on, all of the EMCE window banks remain within their predicted one standard deviation confidence bounds, while the EKF state-estimate error diverges far outside of its predicted confidence bound in both relative lateral position and velocity. Moreover, the confidence bound of all EMCE window banks for relative lateral position crosses inside that of the EKF confidence bound. This is possible because the EMCE covariance is a function of the measurement itself, and can adjust dynamically to its measurement history.

Figure 8.8 shows a magnified illustration of the state error in the relative position



Figure 8.7: EKF and EMCE estimation errors (solid) and one-sigma standard deviation bounds (dashed).

given in Fig. 8.7. The performance of the EKF and EMCE is similar when the noise is close to Gaussian, as seen in Figs. 8.6 and 8.8 for the first 80 time-steps. Moreover, there does not seem to be a large performance increase from using a sliding window-bank of the last four measurements to a sliding window bank of the last eight measurements. All window bank sizes perform competitively to the EKF in the first 80 time-steps, and all window banks are seen to greatly outperform the EKF once any heavy-tailed noise is observed. This implies that, with only a very slight loss of estimation accuracy, one could trade-off larger window bank sizes for faster execution speeds. We see that the practical advantage of the EMCE for nonlinear systems is that the estimator stays robust to heavy-tailed noises and outliers in the measurement (or the process), which could otherwise cause large divergences or even failures to an EKF, as demonstrated by Fig. 8.7.

The execution time for the EMCE window banks are simply those presented in Table 8.5, as this problem is also three-state. The results showed clearly that the sliding window approximation is real-time capable for three state estimation problems. For powerful GPUs such as the NVIDIA V100, the estimator is able to achieve well over 90 hertz



Figure 8.8: Close-up of the top subplot in Fig. 8.7 : relative lateral position estimation error.

for window sizes of eight or less. For estimates conditioned upon nine or ten measurements, respectively, the execution time suffers due to the large number of terms required to evaluate the characteristic function. However, we see that within the scope of the problem presented, the estimation performance is perfectly sufficient for window banks of size four or greater. On a more reasonably priced GPU, such as a Geforce GTX 1060, the estimator can still achieve admirable execution rates for windows of size seven or less. Using a window of 4 would yield a Hz-rate of over 1000 on the V100 GPU.

#### Monte Carlo Simulation

The performance of the EMCE against the EKF is now presented for five Monte Carlo simulations, each of 9000 trials. For all experiments here, the homing missile guidance law  $a_{p_k}$  of (8.9) is used. For proper comparison between the estimators and to keep the focus on estimation performance, the guidance law is implemented as a function of the underlying simulation state and not the state estimates. Consequently, the guidance law helps mitigate the nonlinearity in the measurement model by keeping the line-of-sight bearing angle  $\theta_k$  near zero and in effect, keeps the EKF from diverging dramatically, as was discussed in section 8.2.2.

Due to the heavy-tailed measurement noise, ensemble averages of the state variances will diverge as the number of trials tends toward infinity. Instead, the criterion used is the geometric mean of the state error, which is known to exist when the noises are heavy-tailed [7]. For the five Monte Carlo experiments, measurement noise was generated for each experiment using the underlying  $S-\alpha-S$  noise generator parameterized by  $\alpha =$  $\{2.0, 1.7, 1.5, 1.3, 1.0\}$ , respectively. The measurement noise statistics of the EMCE and EKF, however, were kept equal to those of section 8.2.2 for  $\alpha = 1.7$ . Effectively, by keeping the EKF and EMCE statistics constant across each experiment (but varying the measurement noise severity), we evaluate the robustness of the estimators to uncertainty in the model of the noise statistics. The range of  $\alpha$  values chosen is motivated by the fact that radar in environmental clutter has been observed to follow  $S-\alpha-S$  distributions with  $\alpha$  values as low as 1.1.



Figure 8.9: Monte-Carlo simulations of 9000 trials for  $\alpha = \{2.0, 1.7, 1.5, 1.3, 1.0\}$ . The performance criterion is the geometric mean error. EMCE window bank of seven is in blue, against the EKF (green).

Figure 8.9 illustrates the performance of the EKF against the EMCE for a modest window bank size of seven. We see that, remarkably, there is almost no change observed in the performance index for the EMCE in the relative position and velocity as  $\alpha$  is lowered from 2.0 to 1.0, while the EKF degrades dramatically. Unsurprisingly, the EKF is the superior estimator in the pure Gaussian noise environment, whereas the EMCE is superior for all heavy-tailed noise levels tested in relative position and velocity. We conclude that the Cauchy estimator appears to either be or close to a minimum variance estimator over the class  $\alpha \in [1, 2]$ . In this sense, the EMCE is considered to be robust.

Both estimators struggle to estimate the acceleration state of the target, for which the Monte Carlo trials do not produce convergent results for heavy-tailed noise  $\alpha < 2.0$  as they do for the relative position and velocity. We suspect that tens of thousands more trials would be necessary to produce convergent results for target acceleration in Fig. 8.9, which is estimated poorly, regardless.

Figures 8.7 and 8.9 suggest that the net benefits of the EMCE for heavy-tailed noise environments are: 1) near identical performance relative to the EKF is expected during periods of time when the noise is effectively Gaussian, 2) superior performance is expected when the noise becomes more volatile than the Gaussian distribution suggests, and 3) the EMCE is expected to stay robust and help safeguard against modes of failure that can otherwise occur during nonlinear system estimation, as demonstrated by step 84 of Fig. 8.7. Moreover, unlike the Kalman filter, Fig. 8.9 implies that regardless of how heavy-tailed the noise characteristics can become, the mathematical structure of the Cauchy estimator allows it to remain resilient to outliers of varying magnitudes and with negligible impact on its performance index.

# 8.3 A Five-State Low Earth Orbit Satellite Simulation in Heavy-Tailed Noise

# 8.3.1 Formulation

A simulation of a low earth orbit (LEO) satellite following a circular orbit around the Earth is now presented. The example here follows closely to the simulation dynamics presented in [12], however, the proposed experiment is different. The simulation dynamics can be written as

$$\frac{d^2}{dt^2}\vec{\mathbf{r}} = -\frac{\mu}{r^3}\vec{\mathbf{r}} - \frac{1}{2}C_D\frac{A}{m}\rho\left(1 + \frac{\delta\rho}{\rho}\right)v_r\vec{\mathbf{v}}_r \tag{8.10}$$

$$\frac{d}{dt}\left(\frac{\delta\rho}{\rho}\right) = -\frac{1}{\tau}\left(\frac{\delta\rho}{\rho}\right) + w_p \tag{8.11}$$

where  $\vec{\mathbf{r}}$  is the position of the center of gravity of the satellite, expressed in a geocentric and celestially referenced system of coordinates,  $\vec{\mathbf{v_r}}$  is the velocity of the satellite in a terrestrially fixed geocentric system of coordinates that is aligned with Earth's rotational axis, and  $r = ||\vec{\mathbf{r}}||, v_r = ||\vec{\mathbf{v}}_r||$  are the euclidean norm of these quantities, respectively [12]. In (8.10),  $\mu$  is the universal gravitation constant,  $C_D$  is the coefficient of drag for the satellite, A is the area of the satellite in the plane normal to  $\vec{\mathbf{v}}_r$ , m is the mass of the satellite,  $\rho$  is the nominal atmospheric density of the satellite at its orbital height, and  $\delta\rho$ represents a change to the nominal density. In (8.11), the state  $\frac{\delta\rho}{\rho}$  is taken to be the change in the atmospheric density over the nominal value and  $\tau$  is a first-order time constant. In (8.11),  $w_p$  is an additive random noise, which was shown in [12] from experimental data to be non-Gaussian and best modelled by an  $S-\alpha-S$  distribution with  $\alpha = 1.3$ . Hence, it is seen that changes in dynamic pressure occur suddenly and impulsively during low earth orbit.

In [12], an EKF and a scalar Cauchy estimator work in unison, where the scalar Cauchy estimator is given the discretized dynamics of (8.11) and the EKF is given the discretized dynamics of (8.10). GPS range measurements are provided to the EKF, using the simplified measurement model of

$$z_k^i = ||\vec{\mathbf{r}} - \vec{r_i}|| + v_k, \quad i \in [1, .., m]$$
(8.12)

where  $z_k^i$  is the range from the LEO satellite to another satellite *i* at time step *k* and out of *m* satellites that provide range measurements. Above,  $v_k$  is the measurement noise of the GPS, which is assumed Gaussian. In [12], the scalar Cauchy estimator uses a virtual drag acceleration sensor measurement that is computed using the EKF's velocity estimate of the satellite as

$$d = \frac{1}{2} C_D \frac{A}{m} v_r^2 \rho \left( 1 + \frac{\delta \rho}{\rho} \right) \tag{8.13}$$

where  $v_r$  is the (norm of the) conditional mean of the EKF's velocity estimate. The scalar Cauchy estimator then supplies its estimate of the conditional mean and variance of  $\frac{\delta\rho}{\rho}$  to the EKF to discipline its mean and covariance estimates for (8.10) using the Kalman-Schmidt recursion (see [12]). Hence, each estimator bootstraps off the other. It was seen that disciplining the EKF with estimates from the scalar Cauchy estimator
aided in estimating the atmospheric density of the satellite during its orbit. At the time [12] was published, only the scalar cauchy estimator had a form that was computationally tractable.

Now that the MCE is capable of estimation over long horizons, non-linear dynamics and non-linear measurement models, the simulation conducted in [12] can be done without bootstrapping one estimator off of the other and *solely* through the GPS measurements, without (8.13). By conducting the full state estimation problem using the EMCE, the estimator's performance when processing both nonlinear dynamics and measurement models can now be shown. It is expected that all states should be estimated well during volatile changes in atmospheric density.

As the orbit is assumed circular, we pose this problem using five states, with two position states, two velocity states, and atmospheric density. Both the EMCE and the EKF are formulated through discretizing the nonlinear transition model of (8.10) and (8.11) into the structure of (3.24). Specifically, the EKF and EMCE form the linear dynamics

$$\Phi_k = \sum_{i=0}^{L} \left( \nabla_x f(\hat{x}_k) \right)^i \frac{\Delta^i}{i!}$$
(8.14)

$$\Gamma_k = \int_0^{\Delta} \Phi_k(\tau) \Gamma d\tau, \quad \Gamma = [0, 0, 0, 0, 1]^T$$

$$= \int_0^{\Delta} \sum_{i=1}^{L} (\nabla_i f(\hat{x}_i))^i \tau^i \Gamma d\tau$$
(8.15)

$$= \int_{0}^{L} \sum_{i=0}^{L} (\nabla_{x} f(\hat{x}_{k}))^{i} \frac{\Delta^{i+1}}{i+1!} \Gamma$$
$$= \sum_{i=0}^{L} (\nabla_{x} f(\hat{x}_{k}))^{i} \frac{\Delta^{i+1}}{i+1!} \Gamma$$
$$H_{k+1} = \nabla_{x} h(\bar{x}_{k+1}), \quad \bar{x}_{k+1} = f(\hat{x}_{k})$$
(8.16)

where in (8.14), a power series is used to form the discrete time transition dynamics. The power series uses L = 2, and  $\Delta = 60$  seconds, as was specified in [12]. The superscript *i* on the gradient of  $f(\hat{x}_k)$  denotes the matrix power of the gradient of (8.10) and (8.11) with respect to the state estimate  $\hat{x}_k \in \mathbb{R}^5$ . The discrete time control matrix  $\Gamma_k$  of (8.15) is formed by integrating the power series of (8.14) and multiplying by the continuous time control matrix  $\Gamma$ . Equation (8.16) is the linearized measurement model about the propagated state  $\bar{x}_{k+1}$  by passing the state estimate  $\hat{x}_k$  through the nonlinear dynamics. As  $\Delta = 60$  seconds is a large quantity of time, a runge-kutta integration scheme is used to propagate the state through the nonlinear dynamics using half second sub-intervals until 60 seconds has passed. The EKF uses  $\Phi_k, \Gamma_k, H_k$  to propagate and update the covariance estimates in its traditional fashion, while the EMCE uses these matrices as given by algorithm 13. Both estimators use (8.12) as their measurement model, with two range measurements given per time-step.

#### 8.3.2 Numerical Results

The LEO simulation sets the nominal orbital height to r = 200 km, A = 64m,  $C_D = 2.0$ ,  $\tau = 21600$ , m = 5000 kg,  $\rho = 2.541 e^{-10}$ , and the time between measurements is  $\Delta = 60$ . The process noise parameter  $\beta$  was set to 0.0013 for  $\Delta = 60$ , based on the observed  $S^{-1.3}-S$  fitting of atmospheric density in [12]. The discrete-time process noise for the EMCE follows this quantity and the EKF fits to this noise using (1.6). The GPS noise is assumed Gaussian with 2.0m standard deviation, and two GPS range measurements were used per time step. The EKF follows this quantity, while the EMCE fits its measurement noise parameter to this quantity via (1.6). The period of the orbit is 88 minutes, and the simulation is run over roughly 3.5 orbits, i.e., 300 time steps. For the EMCE, the sliding window approximation with a bank of five windows is used, each window processing both GPS range measurements per time step.

Depicted in Fig. 8.10 are the measurement and noise realizations used in the simulation. The range measurements (top) are given to both estimators as the LEO satellite makes its orbit. The Gaussian measurement noise realization (middle) corrupts these range measurements. The process noise realization (bottom) is drawn from the  $S-\alpha$ -Sdistribution with  $\alpha = 1.3$  and a scaling parameter that uses (1.6) to convert  $\beta$  from  $\alpha = 1$ to 1.3, in the least squares sense. Three large jumps in atmospheric density occur in the first 100 time steps. The first and largest jump is around k = 20, which will change the atmospheric density from its nominal value by 150% (i.e.,  $\frac{\delta\rho}{\rho} \approx 1.5$ ). At around k = 60, another jump appears and the density is reduced, and again around k = 100. After which, there are no large changes in density. In [12], changes up to 600% for the nominal density value were seen to occur, and therefore the realization here is not considered extreme, by any accounting.



Figure 8.10: Top: simulated measurement ranges provided to the estimators. Middle: realization of additive measurement noises. Bottom: realization of additive process noise forcing atmospheric density.

Figure 8.11 shows the state errors of the EKF and EMCE for this realization. Although the GPS measurements have a standard deviation of 2m, the effect of underestimating density is apparent by observing the EKF's large state errors in position and velocity when jumps occur. The EKF very slowly integrates these volatile changes in atmospheric density. During the jumps, the EKF suffers from 'filter smugness', meaning, the Kalman gain of the EKF is tailored to trust the process dynamics much more than its sensor measurements. Therefore, large (and unexpected) changes in density are



Figure 8.11: EKF estimation errors (dashed green) and one-sigma standard deviation bounds (dashed magenta). EMCE estimation errors (blue) and one-sigma standard deviation bounds (red).



Figure 8.12: EKF and EMCE state estimates (green, blue), respectively, and true state history of change in atmospheric density (red).

dismissed, leading to large errors during the EKF's time propagation step (using the runge-kutta integrated (8.10) and (8.11)) as well as in the EKF's covariance calculations (using the linearized system matrices (8.14) to (8.16)). This makes sense, as noises that are hundreds of deviations away from the mean are infinitely rare under the Gaussian assumption, but do occur when sampling from a heavy-tailed  $\alpha = 1.3$  distribution.

Shown in Fig. 8.12 are the state estimates of atmospheric density for the EKF and the EMCE, as well as the true simulated state realization of  $\frac{\delta\rho}{\rho}$ . We see in Fig. 8.12 and Fig. 8.11 (bottom) that after a half orbital period (i.e., k = 20 to k = 60), the EKF has still not converged to the true density state. Large errors are produced at k = 60once more in position and velocity after atmospheric density jumps again. It is clear the one-sigma confidence bound of the EKF greatly underestimates its state error in all five of the states throughout the entire simulation.

Compared to the EKF, Fig. 8.11 shows that no large errors in position and velocity are accrued for the EMCE. It is clear that the EMCE is far superior in estimating position and velocity during impulsive jumps in atmospheric density. Just as important is the fact that the EMCE's one-sigma confidence bound reflects the observed state errors of all states, whereas the EKF does not. Moreover, the EMCE does not suffer from the same 'smugness' the EKF experiences. To summarize, the EMCE accurately estimates the position and velocity of the satellite over the entire duration of the simulation, and with covariance estimates that bound the state errors appropriately. As was discussed in [12], the predictive task of estimating the probability of collision with other LEO satellites is an important extension to the presented estimation problem, and consequently, when orbital adjustments of the LEO satellite should take place. Because the EMCE's onesigma confidence bound of Fig. 8.11 appropriately hypothesizes the state error, the state and covariance estimates of the EMCE could prove to be useful for this task.

It is seen from Fig. 8.12 and the bottom subplot of Fig. 8.11 that atmospheric density (blue) is estimated somewhat more appropriately by the EMCE during jumps in density, although, the estimates are bouncy and non-smooth. We suspect that this is a result of the sliding window approximation using only a window size of 5. Although not depicted, it was seen from experiments that the smoothness of all estimates of the EMCE increased dramatically when the window size was expanded from three to five. As atmospheric density is a difficult quantity to observe, we suspect that recovering the atmospheric density estimate using GPS measurements of 2m standard deviation is challenging when using only a window size of 5. Future work should consider window sizes of up to 8. Although the EMCE's estimates in density are somewhat sub-optimal, the EMCE is seen to track the changes in atmospheric density. From time steps k = 150 to k = 300, the estimates in atmospheric density using the EKF are superior, as the process noise realization is much closer to Gaussian.

Furthermore, it should be noted that the EMCE's position and velocity error also experience slightly non-smooth behavior. This is induced by the non-smooth atmospheric density estimates, although less pronounced. Determining how to smooth out the density estimates of the EMCE is left to future work. It is clear, however, that position and velocity estimates are far superior to that of the EKF even with the observed 'chattering' in the state estimates. If the density estimates could be smoothed, it is suspected the state estimates for position and velocity would improve further. Regardless, it appears as though the position and velocity estimates remain resilient to both the underlying jumps in density and the sub-optimal density estimates. Even if the aforementioned setbacks were to remain unresolved, it is suspected the EMCE would be a good estimator for predictive tasks such as the probability of collision, nevertheless.

#### Estimation of density in a lower measurement noise

Here, the (slightly unrealistic) scenario is considered where the GPS sensor measurements now have centimeter-level accuracy. This is done to illustrate that the EMCE using the sliding window approximation is perfectly capable of smoothly estimating density, if given a sensor measurement that improves the observability of atmospheric density. For simplicity, this is done here by scaling down the measurement noise. The simulation realization presented in Fig. 8.10 is unchanged, with the exception that the standard deviation of the measurement noise is lowered from 2.0m to 0.02m. Both estimators are re-calibrated for this new measurement noise level. Figure 8.13 illustrates the state estimates of the change in atmospheric density  $\frac{\delta \rho}{\rho}$  for the EKF and EMCE for this new scenario. Shown in red is the true simulation state history, which is identical to that presented in Fig. 8.12. It is clear that the EMCE now estimates the density smoothly and detects the density jumps immediately. Even in this low-noise setting, the EKF takes many steps to re-converge after the density is seen to jump. For reference, at  $k \approx 20$ , the EKF still takes about 30 time-steps for the density estimates to reconverge (at  $k \approx 50$ ) after the first jump, while the EMCE needs only several.

Figure 8.14 illustrates the state errors of the atmospheric density for both the EKF and EMCE. State errors for position and velocity are not given as they follow very closely to those in Fig. 8.11, with the EMCE greatly outperforming the EKF in position and



Figure 8.13: EKF and EMCE state estimates (green, blue), respectively, and true state history (red) of change in atmospheric density for low noise simulation.

velocity. Not only does the EMCE estimate density well, but the predicted one-sigma confidence bound of the EMCE does well at encapsulating the state error of the density estimates. The EKF's one-sigma confidence bound, however, is nearly useless.



Figure 8.14: EKF estimation errors (green dashed) and one-sigma standard deviation bounds (magenta dashed). EMCE estimation errors (blue) and one-sigma standard deviation bounds (red).

The results here indicate that the EMCE is perfectly capable of estimating atmospheric density using a higher fidelity measurement and a window of 5. Based on these insights, several possible directions for future work could be explored. First, the results above indicate the five-state simulation could benefit from adding another sensor measurement, such as one derived from a high-fidelity accelerometer. Adding another measurement that increases the observability of density could prevent extending the window size. The second direction is to increase the window size and allow the EMCE to condition its estimates on a larger measurement history. This was not done here due to memory limitations (>8GB), however, a CPU (or multiple GPU) computer with 64GB to 128GB of (device) RAM can easily handle window sizes of up to 8.

#### Chapter 9

#### **Conclusions and Future Work**

This dissertation proposes methods to replace the computationally intractable and backwardrecursive characteristic function of the original MCE algorithm with a non-backward recursive (compressed) and computationally streamlined form. In addition to the theoretical contributions made by this dissertation, the sub-routines that comprise the newly compressed characteristic function were analyzed for the purpose of distributing their computation onto GPU devices. The net sum of these contributions leads to an estimator that is capable of real-time performance and real-world engineering applications. Furthermore, the MCE is seen to be very robust within heavy-tailed noise environments. The contributions that were presented in this dissertation, as well the future work remaining for these contributions are formally enumerated below.

1. The compressed characteristic function of the MCE presented reduced the memory burden of the estimator by allowing terms of previous estimation steps to be discarded. By compressing the backwards recursive component of the characteristic function, all similar terms can now be combined together at each estimation step. This was seen to remove over 99% of terms after several estimation steps, when compared to the original formulation. The technique presented in chapter 3 was reliant on a cell-enumeration algorithm to determine the sign vectors of central hyperplane arrangements. The cell enumeration algorithm (Inc-Enu) was then refactored into a GPU driven one, allowing for faster construction of these sign-vectors. Cell enumeration is an expensive procedure, nevertheless. It is interesting to note that the hyperplane arrangements of the MCE are generated in a deterministic way by the system dynamics { $\Phi, \Gamma, H$ }. There is likely a more elegant way of determining the sign-vectors. Future work should explore how to enhance Inc-Enu, or all together replace this sub-routine. Such a method should explicitly consider how the child arrangements generated by (3.8a) geometrically change their parent arrangement. Furthermore, insights similar to those made in [22] regarding a-priori knowledge of term coalignment and term reduction would be very useful for the time-varying and nonlinear system cases, if they can be made. Lastly, approximations of the characteristic function, such as those in section 3.7.1 should be further studied. It is likely that there are more clever ways to discard terms and approximate the characteristic function.

- 2. An efficient method was proposed to construct the parameter vectors used to compress the characteristic function, without the need of a cell enumeration and system of equations solver. This method, proposed in chapter 4, was seen to be over twice as fast as the method of chapter 3. However, this method could be much faster if a more careful approach to using the hash-tables was taken. Instead of using a single hash table to store the child g-values (i.e, (3.20)), if all children (of the same parent) shared an additional hash table to cache the left and right hand numerators of (3.20), many more cache hits would be seen. The speed up over the current implementation of chapter 3 could be as large as twice the number of children-per-parent. This was not attempted, however, as the programming complexity to keep track of these children after measurement coalignment was difficult. This is because after measurement coalignment, some child terms coalign several hyperplanes, some a single, and others none. Therefore, they sit within different memory locations in the array of array data structure used to group terms of the same arrangement size. Future work should certainly consider this enhancement.
- 3. The sliding window approximation was extended to multivariate systems in chapter 5. This allows a window that was reset to recreate the estimate of the window

conditioned on the most (i.e, the window size) measurements over a single estimation step. As the number of terms grow each estimation step, the sliding window is used to fix the computational burden of the estimator so that it can be used for arbitrarily long estimation horizons. Due to the insights given in chapter 5, it is now possible to run the estimator for arbitrarily long horizons in the multivariate case, while also initializing each window with the best information available. Future work should consider how this windowing technique can be better applied to compute clusters that are distributed over multiple machines. A prototype for this case was built using socket communication, however, this prototype is likely far from optimal. In the case of very large system dynamics, the estimator will require a large amount of (GPU) memory. This may not be available on a single machine, even if the machine has multiple GPUs. In these cases, coordination between the windows and communication delay may significantly hinder the possible run-time performance of the estimator.

- 4. GPU programming was seen to significantly increase the run-time performance of the estimator. The MCE distributes its computation over the compute cores of the GPU, for each sub-routine of the compressed characteristic function, as was seen in chapter 6. Several optimizations to the current GPU-driven MCE could be made. First, each MCE used by the sliding window approximation is run on a single GPU. Hence, one GPU is used per characteristic function. This is usually sufficient, however, if each MCE utilized multiple GPUs for estimation steps involving large numbers of terms, the execution time could be increased further. It is seen by profiling the application that the term-reduction and α-vector generation sub-routines require the majority of the compute time, while the others make up very little. If further optimizations are made to the GPU-based application, attention should be focused to these sub-routines. The current GPU code is made available for academic use at https://github.com/natsnyder1/KingCauchy. Matlab and Python wrappers for the C/CUDA-C programs are under development.
- 5. The MCE was extended to handle non-linear system dynamics in chapter 7, similar

to the formulation used in the EKF. Three simulations were provided in chapter 8 to showcase the performance of the MCE in a range of noise environments and state dimension. By far the most interesting result is that depicted in section 8.2, where the MCE performance across a range of (unmodeled)  $S-\alpha-S$  noises is seen not to degrade. In this sense, the estimator is robust. The provided experiments indicate that the MCE is likely a good candidate for any application whose process or measurement noise density is within the range of  $\alpha \in [1, 2)$ . Above all other recommended future work, efforts should focus on finding exciting applications where the environmental noise is identified to be heavy-tailed. Such applications would benefit greatly from the MCE. It is likely that many of these applications exist in the defense sector, in space applications, and in the field of quantitative finance.

## Appendix A

# Appendix

## A.1 Least Squares fit of Symmetric- $\alpha$ -Stable Characteristic Functions

This appendix gives the derivation to fit the Cauchy pdf to that of a Gaussian pdf in the least squares sense. The derivation can be done using either pdfs of characteristic functions. Both yield the same result. The solution presented here uses characteristic functions. Only for Gaussian and Cauchy pdfs can an analytic solution be found. When fitting a pdf that is not Gaussian or Cauchy, a numerical approach must be taken instead.

Plancherel's theorum states that for any two scalar and real-valued functions f(x)and g(x),

$$\int_{-\infty}^{\infty} f(x)g(x)dx = \int_{-\infty}^{\infty} \hat{f}(\zeta)\hat{g}(\zeta)d\zeta$$
(A.1)

$$\hat{f}(\zeta) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi\zeta x}d\zeta$$
(A.2)

$$\hat{g}(\zeta) = \int_{-\infty}^{\infty} g(x) e^{-i2\pi\zeta x} d\zeta$$
(A.3)

which is to say that the integral of their product is equal to the integral of the product of their frequency spectrums. Above,  $\zeta$  is the spectral variable and *i* is the imaginary number. Letting  $\eta = 2\pi\zeta$  and j = -i, we can restate either (A.2) and (A.3) as

$$\Phi_x^f(\eta) = \hat{f}\left(\frac{-\eta}{2\pi}\right) = \int_{-\infty}^{\infty} f(x)e^{j\eta x}dx \tag{A.4}$$

where  $\Phi_x^f(\eta)$  is the characteristic function representation of f(x). Plugging in the transform  $d\eta = \frac{\zeta}{2\pi}$  to (A.1) we arrive at the required starting form

$$\int_{-\infty}^{\infty} f(x)g(x)dx = \frac{1}{2\pi} \int_{-\infty}^{\infty} \Phi_x^f(\eta)\Phi_x^g(\eta)d\eta.$$
(A.5)

To fit a Gaussian pdf g(x) to a Cauchy pdf f(x), or a Cauchy to a Gaussian pdf in the least squares sense, the expression

$$\int_{-\infty}^{\infty} (g(x) - f(x))^2 dx = \int_{-\infty}^{\infty} \left( g(x)^2 - 2f(x)g(x)dx + f(x)^2 \right) dx$$
(A.6)

is first re-expressed in its characteristic function form as

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} \left( \left( \Phi_x^g(\eta) \right)^2 - 2\Phi_x^f(\eta) \Phi_x^g(\eta) + \left( \Phi_x^f(\eta) \right)^2 \right) d\eta, \tag{A.7}$$

The characteristic function of the (zero mean) Gaussian is  $\Phi_x^g(\eta) = e^{\frac{-\sigma_g^2 \eta^2}{2}}$  and the characteristic function of the (zero median) Cauchy is  $\Phi_x^f(\eta) = e^{-\sigma_c |\eta|}$ . Minimizing (A.7) w.r.t  $\sigma_g$  yields our desired least squares expression to fit a Gaussian to a Cauchy pdf, given by

$$\underset{\sigma_g}{\operatorname{argmin}} \frac{1}{2\pi} \int_{-\infty}^{\infty} \left( e^{-\sigma_g^2 \eta^2} - 2e^{\frac{-\sigma_g^2 \eta^2}{2} - \sigma_c |\eta|} + e^{-2\sigma_c |\eta|} \right) d\eta.$$
(A.8)

Taking the derivative of (A.8) and setting it equal to zero yields

$$0 = \frac{1}{2\pi} \int_{-\infty}^{\infty} \left( 2\eta^2 \sigma_g e^{\frac{-\sigma_g^2 \eta^2}{2} - \sigma_c |\eta|} - 2\eta^2 \sigma_g e^{-2\sigma_g^2 \eta^2} \right) d\eta$$
  
= 
$$\int_{-\infty}^{0} \eta^2 e^{-\left(\frac{\sigma_g^2 \eta^2}{2} - \sigma_c \eta\right)} d\eta + \int_{0}^{\infty} \eta^2 e^{\frac{-\sigma_g^2 \eta^2}{2} - \sigma_c \eta} d\eta - \int_{-\infty}^{\infty} \eta^2 e^{-\sigma_g^2 \eta^2} d\eta.$$
(A.9)

Let us deal with the three integral expressions of (A.9) separately. For the first integral, completing the square of the exponent yields

$$\left(\frac{\sigma_g^2 \eta^2}{2} - \sigma_c \eta\right) = \frac{\sigma_g^2}{2} \left(\eta - \frac{\sigma_c}{\sigma_g^2}\right)^2 - \frac{\sigma_c^2}{2\sigma_g^2}.$$

Substituting  $u = \eta - \frac{\sigma_c}{\sigma_g^2}$  and  $\theta = \frac{\sigma^2}{2}$ , the integral can be written, evaluated and simplified as

$$\int_{-\infty}^{0} \eta^{2} e^{-\left(\frac{\sigma_{g}^{2}\eta^{2}}{2} - \sigma_{c}\eta\right)} d\eta = \int_{-\infty}^{0} \eta^{2} e^{-\left(\frac{\sigma_{g}^{2}}{2}\left(\eta - \frac{\sigma_{c}}{\sigma_{g}^{2}}\right)^{2} - \frac{\sigma_{c}^{2}}{2\sigma_{g}^{2}}\right)} d\eta \tag{A.10}$$

$$= e^{\frac{\sigma_c^2}{2\sigma_g^2}} \left[ \int_{-\infty}^{\frac{-\sigma_c}{\sigma_g^2}} u^2 e^{-\theta u^2} du + \frac{2\sigma_c}{\sigma_g^2} \int_{-\infty}^{\frac{-\sigma_c}{\sigma_g^2}} u e^{-\theta u^2} du + \frac{\sigma_c^2}{\sigma_g^4} \int_{-\infty}^{\frac{-\sigma_c}{\sigma_g^2}} e^{-\theta u^2} du \right]$$
(A.11)

$$= e^{\frac{\sigma_c^2}{2\sigma_g^2}} \left[ \frac{1}{4} \sqrt{\frac{8\pi}{\sigma^6}} \left( 1 - \operatorname{erf}\left(\frac{\sigma_c}{\sqrt{2}\sigma_g}\right) \right) - \frac{\sigma_c}{\sigma_g^4} e^{\frac{-\sigma_c^2}{2\sigma_g^2}} + \frac{\sigma_c^2}{\sigma_g^4} \left[ \frac{1}{2} \sqrt{\frac{2\pi}{\sigma_g^2}} \left( 1 - \operatorname{erf}\left(\frac{\sigma_c}{\sqrt{2}\sigma_g}\right) \right) \right] \right].$$
(A.12)

Fortunately, the middle integral of (A.9) evaluates to the same quantity as (A.12). The right integral of (A.9) simply evaluates to

$$\int_{-\infty}^{\infty} -\eta^2 e^{-\sigma_g^2 \eta^2} d\eta = -2 \int_0^{\infty} \eta^2 e^{-\sigma_g^2 \eta^2} d\eta = \frac{-\sqrt{\pi}}{2\sigma_c^3}.$$
 (A.13)

Adding the evaluated integrals of (A.9) together, letting  $\theta = \frac{\sigma_c}{\sqrt{2}\sigma_g}$ , and fully simplifying, the resultant expression is

$$0 = e^{\theta^2} \left( 1 + 2\theta^2 \right) \left( 1 - \operatorname{erf}(\theta) \right) - \frac{2}{\sqrt{\pi}} \theta - \frac{1}{2\sqrt{2}}$$
(A.14)

Although (A.14) is a closed-form solution to (A.9),  $\theta$  must be recovered through a numerical search. The value of  $\theta$  which sets (A.14) equal to zero is 0.508783. Now, we see that the ratio  $\frac{\sigma_g}{\sigma_c} = 1.3898$ , which minimizes (A.8) in the least squares sense, w.r.t  $\sigma_g$ . If the analysis is repeated by minimizing  $\sigma_c$  instead, the result will be  $\frac{\sigma_c}{\sigma_g} = \frac{1}{1.3898}$ .

#### A.2 Equating Gauss-Markov to Poisson Telegraph Statistics

Here we equate the second order statistics of the Gauss-Markov process for the target acceleration model of (8.2) to that of a telegraph wave which has Poisson distributed switching times. The target acceleration is given by

$$\dot{a}_T = -\frac{1}{\tau}a_T + w_{a_T} \tag{A.15}$$

with first order lag parameter  $\tau > 0$ ,  $w_{a_T}$  is white noise with  $\mathbb{E}[w_{a_T}] = 0$ ,  $\mathbb{E}[w_{a_T}(\sigma)w_{a_T}(\nu)] = W\delta(\sigma - \nu)$  where  $\delta(\sigma - \nu)$  is the Dirac delta function and W is the spectral density. The steady state autocorrelation  $R_{a_T a_T}(t, s) = \mathbb{E}[a_T(t)a_T(s)]$  and discrete time process noise statistics easily follow as

$$R_{a_T a_T}(t,s) = \mathbb{E}[a_T^2] \Phi(t-s) = \mathbb{E}[a_T^2] e^{-\frac{1}{\tau}|t-s|}$$
  
=  $\frac{\tau}{2} W e^{-\frac{1}{\tau}|t-s|},$  (A.16)

$$w_k \sim \mathcal{N}\left(0, \frac{\frac{2}{\tau}\mathbb{E}[a_T^2]\delta_{kl}}{\Delta}\right),$$
 (A.17)

where  $w_k$  is the discrete process noise where division by  $\Delta$  approximates the Dirac delta function and  $\delta_{kl}$  is the Kronecker delta function.

Now we turn our attention to the second-order statistics of the telegraph wave. Let  $h_T$  be the height of the telegraph and in this model the value  $h_T$  changes sign at random times given by a Poisson probability. We assume that  $h_T(t_0) = \pm h_T$  with probability .5 and  $h_T(t)$  changes polarity at Poisson times. The probability of k sign changes in a time interval of length T, P(k(T)), is  $P(k(T)) = \frac{(\lambda T)^k e^{-\lambda T}}{k!}$  where  $\lambda$  is the rate. The probability of an even number of sign changes in T, P(even # in T), is

$$P(even\# \text{ in } T) = \sum_{\substack{k=0\\k \text{ even}}}^{\infty} \frac{(\lambda T)^k e^{-\lambda T}}{k!} = e^{-\lambda T} \sum_{k=0}^{\infty} \frac{(1+(-1)^k)(\lambda T)^k}{2k!}.$$
 (A.18)

Since  $e^{\lambda T} = \sum_{k=0}^{\infty} \frac{(\lambda T)^k}{k!}$ , then

$$P(even\# \text{ in } T) = e^{-\lambda T} \left[ \sum_{k=0}^{\infty} \frac{(\lambda T)^k}{2k!} + \sum_{k=0}^{\infty} \frac{(-\lambda T)^k}{2k!} \right] = \frac{1}{2} e^{-\lambda T} \left[ e^{\lambda T} + e^{-\lambda T} \right] = \frac{1}{2} \left[ 1 + e^{-2\lambda T} \right]$$
(A.19)

By a similar process,  $P(odd \# \text{ in } T) = \frac{1}{2} \left[ 1 - e^{-2\lambda T} \right]$ . Then the probability that  $h_T(t)$  is positive is

$$P(a_{T}(t) = h_{T}) = P(h_{T}(t) = h_{T}|h_{T}(0) = h_{T})P(h_{T}(0) = h_{T})$$
  
+  $P(h_{T}(t) = h_{T}|h_{T}(0) = -h_{T})P(h_{T}(0) = -h_{T})$   
=  $\frac{1}{2}P(even\# \text{ in } T = [0, t]) + \frac{1}{2}P(odd\# \text{ in } T = [0, t])$   
=  $\frac{1}{2}\left\{\frac{1}{2}\left[1 + e^{-2\lambda t}\right] + \frac{1}{2}\left[1 - e^{-2\lambda t}\right]\right\} = \frac{1}{2}.$  (A.20)

Then, the mean acceleration,  $\bar{h}_T$  is

$$\bar{h}_T = h_T P(h_T(t) = h_T) - h_T P(h_T(t) = -h_T) = 0.$$
 (A.21)

The autocorrelation,  $R_{h_T h_T}(t_1, t_2) = E[h_T(t_1)h_T(t_2)]$  is

$$R_{h_T h_T}(t_1, t_2) = h_T^2 P(h_T(t_2) = h_T(t_1)) - h_T^2 P(h_T(t_2) \neq h_T(t_1))$$
(A.22)  
$$= h_T^2 \frac{1}{2} \left[ 1 + e^{-2\lambda|t_2 - t_1|} \right] - h_T^2 \frac{1}{2} \left[ 1 - e^{-2\lambda|t_2 - t_1|} \right]$$
$$= h_T^2 e^{-2\lambda|t_2 - t_1|}.$$
(A.23)

If  $\tau = \frac{1}{2\lambda}$  and  $\mathbb{E}[a_t^2] = h_T^2$ , then the autocorrelation of the Gauss-Markov process is the same as the random telegraph signal and the means of both are zero. Substituting these quantities into (8.6) yields the intended result.

## Bibliography

- N. N. Taleb, The Black Swan: The Impact of the Highly Improbable, 1st ed. London: Random House, 2008.
- [2] A. Y. Aravkin, J. V. Burke, and G. Pillonetto, "Robust and trend-following student's t kalman smoothers," SIAM Journal on Control and Optimization, vol. 52, no. 5, pp. 2891–2916, 2014.
- [3] R. E. Kalman, "A new approach to linear filtering and prediction problems," Transactions of the ASME-Journal of Basic Engineering, vol. 82, no. Series D, pp. 35–45, 1960.
- [4] J. Speyer, C.-H. Fan, and R. Banavar, "Optimal stochastic estimation with exponential cost criteria," in [1992] Proceedings of the 31st IEEE Conference on Decision and Control, 1992, pp. 2293–2299 vol.2.
- [5] D. Crisan and B. Rozovskii, Eds., The Oxford Handbook of Nonlinear Filtering. Oxford University Press, 2011.
- [6] B. Ristic, S. Arulampalam, and N. J. Gordon, Beyond the Kalman Filter: Particle Filters for Tracking Applications. Artech House Publishers, 2004.
- [7] R. Fonod, M. Idan, and J. L. Speyer, "Approximate estimators for linear systems with additive cauchy noises," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 11, pp. 2820–2827, 2017.
- [8] G. Samorodnitsky and M. S. Taqqu, Stable Non-Gaussian Random Processes: Stochastic Models with Infinite Variance: Stochastic Modeling. Routledge, 1994.

- [9] P. Tsakalides and C. L. Nikias, "Deviation from normality in statistical signal processing: Parameter estimation," A practical guide to heavy tails: statistical techniques and applications, p. 379, 1998.
- [10] S. Naga Divya, G.and Koteswara Rao, "Stochastic analysis approach of extended h-infinity filter for state estimation in uncertain sea environment," *International Journal of System Assurance Engineering and Management*, 2022. [Online]. Available: https://doi.org/10.1007/s13198-022-01682-6
- [11] P. M. Reeves, A non-Gaussian turbulence simulation. Air Force Flight Dynamics Laboratory, Air Force Systems Command, United ..., 1969, vol. 69, no. 67.
- [12] J. R. Carpenter and A. K. Mashiku, "Cauchy drag estimation for low earth orbiters," in AAS/AIAA Space Flight Mechanics Meeting, no. GSFC-E-DAA-TN19723, 2015.
- [13] M. Idan and J. L. Speyer, "Cauchy estimation for linear scalar systems," *IEEE transactions on automatic control*, vol. 55, no. 6, pp. 1329–1342, 2010.
- [14] N. Snyder, M. Idan, and J. L. Speyer, "Distributed computation of a robust estimator based on cauchy noises," in 2021 60th IEEE Conference on Decision and Control (CDC), 2021, pp. 6584–6590.
- [15] J. L. Speyer, M. Idan, and J. Fernández, "The two-state estimator for linear systems with additive measurement and process cauchy noise," in 51st IEEE Conference on Decision and Control (CDC), 2012, pp. 4107–4114.
- [16] M. Idan and J. L. Speyer, "Multivariate Cauchy estimator with scalar measurement and process noises," SIAM Journal on Control and Optimization, vol. 52, no. 2, pp. 1108–1141, 2014.
- B. Mandelbrot, "The pareto-lévy law and the distribution of income," International Economic Review, vol. 1, no. 2, pp. 79–106, 1960. [Online]. Available: http://www.jstor.org/stable/2525289

- [18] N. Snyder, M. Idan, and J. Speyer, "Real-time robust multivariate estimator for dynamic systems with heavy-tailed additive uncertainties (under review)," in *IEEE Journal on Transactions on Automatic Control*, 2023.
- [19] J. M. Chambers, C. L. Mallows, and B. W. Stuck, "A method for simulating stable random variables," *Journal of the American Statistical Association*, vol. 71, no. 354, pp. 340–344, 1976. [Online]. Available: https: //www.tandfonline.com/doi/abs/10.1080/01621459.1976.10480344
- [20] D. Avis and K. Fukuda, "Reverse search for enumeration," Discrete Applied Mathematics, vol. 65, no. 1, pp. 21–46, 1996, First International Colloquium on Graphs and Optimization.
- [21] M. Rada and M. Cerný, "A new algorithm for enumeration of cells of hyperplane arrangements and a comparison with Avis and Fukuda's reverse search," SIAM Journal on Discrete Mathematics, vol. 32, no. 1, pp. 455–473, 2018.
- [22] Y. Bai, "Properties of the multivariate cauchy estimator," 2016.
- [23] N. Duong, M. Idan, R. Pinchasi, and J. Speyer, "A note on hyper-plane arrangements in R<sup>d</sup>," Discrete Mathematics Letters, vol. 7, pp. 79–85, July 2021.
- [24] J. Sanders and E. Edward Kandrot, Eds., CUDA by example: an introduction to general-purpose GPU programming. NVIDIA Corporation, 2011.
- [25] Y. Bai, J. L. Speyer, and M. Idan, "Efficient cauchy estimation via a precomputational technique," in 2016 IEEE 55th Conference on Decision and Control (CDC), 2016, pp. 1171–1178.
- [26] S. Boyd and L. Vandenberghe, Convex Optimization. USA: Cambridge University Press, 2004.
- [27] G. Dantzig, "Linear programming and extensions," in *Linear programming and ex*tensions. Princeton university press, 2016.

- [28] V. Klee and G. J. Minty, "How good is the simplex algorithm," *Inequalities*, vol. 3, no. 3, pp. 159–175, 1972.
- [29] N. Karmarkar, "A new polynomial-time algorithm for linear programming-ii," Combinatorica, vol. 4, pp. 373–395, 12 1984.
- [30] P. M. Vaidya, "An algorithm for linear programming which requires o(((m+n)n2+(m+n)1.5n)l) arithmetic operations," *Mathematical Programming*, vol. 47, pp. 175–201, 1990.
- [31] J. Fernández, J. L. Speyer, and M. Idan, "Stochastic estimation for two-state linear dynamic systems with additive Cauchy noises," *IEEE Transactions on Automatic Control*, vol. 60, no. 12, 2015.
- [32] N. Ploskas and N. Samaras, "Efficient GPU-based implementations of simplex type algorithms," *Applied Mathematics and Computation*, vol. 250, pp. 552–570, 2015.
- [33] A. Gurung and R. Ray, "Simultaneous solving of batched linear programs on a GPU," in ICPE '19: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering. Association for Computing Machinery, 2019.
- [34] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz,
   A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.
- [35] NVIDIA, P. Vingelmann, and F. H. Fitzek, "Cuda, release: 10.2.89," 2020. [Online].
   Available: https://developer.nvidia.com/cuda-toolkit
- [36] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.
- [37] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza, "Dissecting the graphcore ipu architecture via microbenchmarking," arXiv preprint arXiv:1912.03413, 2019.

- [38] J. L. Speyer and W. H. Chung, Stochastic Processes, Estimation, and Control. Society for Industrial and Applied Mathematics, 2008. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898718591
- [39] N. Twito, M. Idan, and J. L. Speyer, "Maximum conditional probability stochastic controller for scalar linear systems with additive cauchy noises," in 2018 European Control Conference (ECC), 2018, pp. 2708–2713.
- [40] D. E. Gustafson and J. L. Speyer, "Design of linear regulators for nonlinear stochastic systems," *Journal of Spacecraft and Rockets*, vol. 12, no. 6, pp. 351–358, 1975. [Online]. Available: https://doi.org/10.2514/3.56987
- [41] A. Bryson and Y. Ho, Applied optimal control: optimization, estimation, and control. Blaisdell Pub. Co., 1969. [Online]. Available: http://books.google.ch/ books?id=k\_FQAAAAMAAJ